VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

DYNAMICKÝ DEKODÉR PRO ROZPOZNÁVÁNÍ ŘEČI

DYNAMIC DECODER FOR SPEECH RECOGNITION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. MICHAL VESELÝ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. PETR SCHWARZ, Ph.D.**
SUPERVISOR

BRNO 2017

Abstrakt

Výstupem této práce je funkční a značně optimalizovaná implementace dynamického dekodéru, která funguje na principu dynamického generování rozpoznávací sítě a dekódování modifikovaným algoritmem Token Passing. Implementované řešení poskytuje srovnatelné výsledky se vzorovým statickým dekodérem z BSCORE (API firmy Phonexia), přičemž přináší výraznou paměťovou úsporu, která umožňuje využití více komplexních jazykových modelů a usnadňuje integraci do mobilních zařízení či dynamické přidávání nových slov do rozpoznávače.

Abstract

The result of this work is a fully working and significantly optimized implementation of a dynamic decoder. This decoder is based on dynamic recognition network generation and decoding by a modified version of the Token Passing algorithm. The implemented solution provides very similar results to the original static decoder from BSCORE (API of Phonexia company). Compared to BSCORE this implementation offers significant reduction of memory usage. This makes use of more complex language models possible. It also facilitates integration the speech recognition to some mobile devices or dynamic adding of new words to the system.

Klíčová slova

dynamický dekodér, jazykový model, akustický model, rozpoznávání řeči, rozpoznávací síť, algoritmus Token Passing, n-gramové pravděpodobnosti

Keywords

dynamic decoder, language model, acoustic model, speech recognition, recognition network, Token Passing algorithm, n-gram probabilities

Citace

VESELÝ, Michal. *Dynamický dekodér pro rozpoznávání řeči.* Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Schwarz Petr.

Dynamický dekodér pro rozpoznávání řeči

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Petra Schwarze Ph.D, a že jsem uvedl všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Michal Veselý
24. května 2017

Poděkování

Rád bych zde poděkoval svému vedoucímu práce Petru Schwarzovi a nadřízenému Jiřímu Nytrovi za jejich odborné rady, cenné připomínky a čas věnovaný mojí práci. Také bych rád poděkoval celému kolektivu ve firmě Phonexia za poskytnutí motivujícího a přátelského pracovního prostředí. Poděkování bych také rád věnoval své rodině a přítelkyni za jejich silnou morální podporu.

Obsah

Kapitola 1

Úvod

Rozpoznávání řeči je vědní disciplínou, jenž si klade za cíl automatický převod mluvené řeči na text. Za několik posledních desítek let došlo v oblasti rozpoznávání řeči k vystřídání mnoha různých trendů a bylo dosaženo velkého pokroku. Současné nejmodernější systémy hojně uplatňují znalosti z oblasti strojového učení (konkrétně *deep learning*) a veledat (anglicky *big data*). Úspěšnost rozpoznávání těchto systému stále roste a pomalu se začíná přibližovat lidské úrovni (viz např. [15]). Z důvodu zvyšující se kvality rozpoznávačů řeči také přibývá možností jejich využití v komerčním sektoru.

S rostoucí robustností jsou však systémy pro rozpoznávání řeči stále více výpočetně náročné. I s neustále rostoucím výkonem počítačů je velmi složité provádět rozpoznávání v reálném čase a systémy obvykle vyžadují velké množství operační paměti, což značně znemožňuje jejich využití v mobilních zařízeních bez neustálého spojení s výpočetním serverem.

Cílem této práce je implementace dynamického dekodéru modifikací již existujícího statického dekodéru, jenž je v současnosti (k prosinci 2016) používán firmou Phonexia (viz [3]) v jejich API BSCORE (Brno Speech Core). Základní verze tohoto dekodéru byla inspirací pro dekodér v open source projektu Kaldi (viz [11]).

Dekodér je jednou ze základních stavebních kamenů každého rozpoznávače řeči a jedná se o jednu z výpočetně nejnáročnějších částí procesu rozpoznávání. Jeho úkolem je nalézt co nejlepší odhad původně řečené sekvence slov na základě akustického a jazykového modelu (popsáno v kapitolách 2.4 a 2.5).

Doposud byl firmou Phonexia používán statický dekodér, u kterého je celá rozpoznávací síť (viz 2.6), složená z n-gramového jazykového modelu a dalších komponent, načtena do operační paměti ještě před samotným procesem dekódování. Tento přístup je však paměťově velmi náročný (rozpoznávací síť zabírá obvykle jednotky GB).

Implementace dynamického dekodéru umožňuje potlačit výše zmíněný problém vysokých paměťových požadavků, jelikož pro dekódování není celá n-gramová rozpoznávací síť uložena v operační paměti. Je zde uložena pouze unigramová rozpoznávací síť a n-gramové pravděpodobnosti jsou řešeny nahrazováním unigramových na základě vhodně navržené vyhledávací struktury obsahující jazykový model (viz kapitola 5).

Díky paměťové úspoře lze lépe integrovat rozpoznávač do mobilních zařízení i bez neustálého připojení k výpočetnímu serveru, dále dynamický dekodér výrazně usnadňuje proces přidávání nových slov do rozpoznávače (což je komerčně velmi žádanou funkcionalitou), umožňuje využití robustnějších n-gramových jazykových modelů, či integraci jazykových modelů založených na rekurentních neuronových sítích. Přičemž využití robustnějších jazy-

kových modelů, či jazykových modelů založených na rekurentních neuronových sítích má potenciál vyšší úspěšnosti rozpoznávání (viz například [9]).

Nevýhodou jsou potenciálně vyšší časové požadavky na výpočet, jelikož n-gramové pravděpodobnosti musí být vyhledány ve vyhledávací struktuře, jejíž vhodná implementace je jednou ze stěžejních částí dobře fungujícího dynamického dekodéru (viz 5.4.1, 5.4.2 a 5.4.3).

V kapitole 2 práce definuje problém úlohy rozpoznávání řeči a zaměřuje se na dílčí části obecného rozpoznávače, konkrétně na proces předzpracování řečového signálu (2.1), extrakci příznaků (2.2), akustický (2.4) a jazykový (2.5) model a z něj vytvářenou rozpoznávací síť (2.6) a závěrečné zpracování výstupu dekódování (2.8). Velký prostor je zde věnován problematice dekódování (2.7), zahrnující použité algoritmy, kritéria, a především je zde zaveden pojem dynamického dekodéru včetně jeho odlišností od dekodéru statického.

V kapitole 3 je detailně vysvětlen způsob, jakým porovnáváme úspěšnost a kvalitu dvou různých systémů rozpoznávání řeči z hlediska časové a paměťové náročnosti a přesnosti rozpoznávání. Zaměřujeme se zde na vyhodnocování systémů (3.1), pro jehož provedení je nutné disponovat skórovacími datovými sadami, které jsou popsané v kapitole 3.2.

Kapitola 4 je věnována sumarizaci použitých metod pro implementaci rozpoznávače řeči vyvíjeného firmou Phonexia a technologickým parametrům rozpoznávače.

V kapitole 5 jsou popsány veškeré dílčí kroky, jež byly provedeny pro dosažení funkční a optimalizované výsledné podoby dynamického dekodéru. Jedná se především o popis nutné modifikace algoritmu Token Passing použitého statickým dekodérem (5.1), přechodu od trigramové rozpoznávací sítě k unigramové (5.2), způsobu náhrady n-gramových pravděpodobností za unigramové pomocí navržené vyhledávací struktury (5.3) a popisu jednotlivých provedených optimalizací vyhledávací struktury uchovávající jazykový model s n-gramovými pravděpodobnostmi (5.4).

Kapitola 6 popisuje experimenty s komplexnějšími jazykovými modely a dosažené výsledky rozpoznávání pro tyto modely.

Kapitola 2

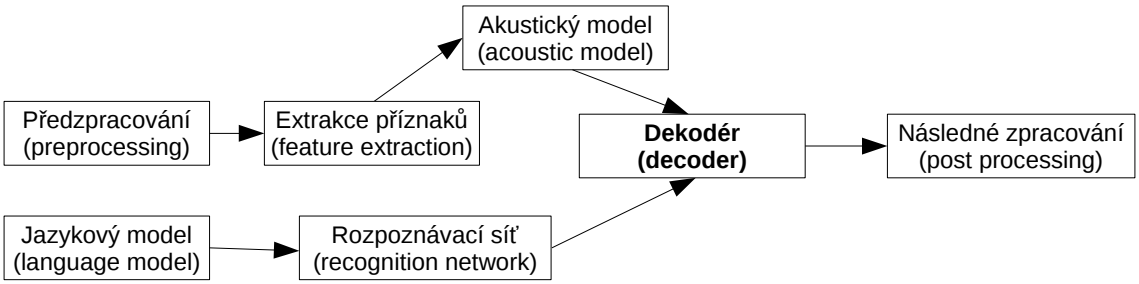
Rozpoznávání řeči

Rozpoznávání řeči je úlohou spočívající v transformaci mluvené řeči na psanou posloupnost slov či fonémů, kde řečí rozumíme posloupnost zvuků. Foném jako abstraktní lingvistická jednotka nepředstavuje konkrétní zvukový segment, ale je asociován s jistými lingvistickými charakteristikami a konfiguracemi hlasového traktu člověka (např. charakteristikami fonému */s/* jsou neznělost, frikativa, při vytváření nekmitají hlasivky, atd.). [12]

Při artikulaci daného fonému pak vzniká vlastní zvuk – akustická realizace fonému. Ačkoliv každému fonému odpovídá jisté základní postavení řečových orgánů, vlastní artikulací dochází k mírným modifikacím této základní konfigurace (vlivem individuality řečníka, spojitým charakterem řeči, atd.). Následkem různých variací základní artikulace fonému je skutečnost, že daný foném může být realizován obrovským množstvím zvuků. [12]

I v případě správného rozpoznání vyřčených fonémů však mohou nastat problémy se správným přepisem do textové formy. Takový problém může nastat u slov stejné výslovnosti ale rozdílného zápisu (např. *být* a *bít*, nebo v angličtině *two* a *to*), případně pokud jsou přepisována slova, jenž jsou složena z jiných slov (např. *kolotoč* a *kolo toč*). Tyto problémy částečně řeší jazykové modelování (viz 2.5).

Výše jsou uvedeny některé základní problémy rozpoznávání řeči. Kvůli těmto problémům, či obecně z důvodu komplexnosti úlohy rozpoznávání řeči, nachází své využití mnohé statistické modely, jež budou popsány dále v rámci popisu jednotlivých komponent obecného schématu rozpoznávače. Obecné schéma rozpoznávače řeči lze vidět na obrázku 2.1.



Obrázek 2.1: Obecné schéma rozpoznávače řeči.

2.1 Předzpracování (preprocessing)

Předzpracování spočívá v načtení vstupního řečového signálu z různých audio formátů do operační paměti a konvertování těchto obecně různých reprezentací signálu do jednotné formy. V rámci předzpracování se tedy provádí **dekódování** v případě kódovaných formátů jako je `mp3` či `opus`, **převzorkování** (anglicky *resampling*) na jednotnou vzorkovací frekvenci (obvykle $8\,kHz$), případně **převod vzorků** na jednotný datový typ (např. $16\,b$ `integer`).

2.2 Extrakce příznaků

Extrakce příznaků (anglicky *feature extraction*) má za úkol transformovat předzpracovaný vstupní řečový signál do více vyhovující podoby pro další zpracování. Jedná se primárně o odstranění irelevantních informací a redukci dimenzionality.

Základem většiny metod analýzy akustického signálu řeči je předpoklad, že se jeho vlastnosti v průběhu času mění pomalu. Tento předpoklad vede na aplikaci metod krátkodobé analýzy, při nichž se úseky řečového signálu vydělují a zpracovávají tak, jako by to byly oddělené krátké zvuky. Tyto mikrosegmenty se nazývají rámce (anglicky *frames*) a jsou reprezentovány krátkými časovými úseky s typickou délkou 20-30 ms. Výsledkem analýzy je pak soubor čísel, které popisují daný rámec. Protože na sebe rámce navazují, dostáváme časové posloupnosti souboru čísel, které popisují daný promluvený celek. [12]

Mezi nejčastěji používané příznaky (*features*) patří Melovské kepstrální koeficienty (anglicky *Mel Frequency Cepstral Coefficient – MFCC*) a odhad základního hlasivkového tónu (anglicky *fundamental frequency*, často značeno jako F_0).

Velmi časté je také klasifikování, zda daný rámec obsahuje řeč či ticho (šum pozadí), přičemž dále jsou zpracovávány pouze řečové rámce. Část rozpoznávače klasifikující rámce na ticho a řeč se nazývá detektor řeči (anglicky *Voice activity detection*, značeno *VAD*). Detektor řeči je součástí většiny řečových technologií, případně jej lze využít jako samostatnou technologii. Nejběžnější implementace VAD je založená na funkci krátkodobé energie signálu (viz rovnice 2.1), případně mohou být použity i jiné již extrahované příznaky, které jsou dále zpracovány například neuronovou sítí. Energii jednoho rámce řeči lze spočítat dle vzorce:

$$E_n = \sum_{k=0}^{N-1} [s(k)w(k)]^2 \tag{2.1}$$

Kde s(k) je vzorek signálu, w(k) je typ okénka (např. Hammingovo – viz [7]) a N je délka rámce.

2.3 Statistický přístup k rozpoznávání řeči

Obecné rozpoznávání řeči je velmi komplexní úlohou. Především pokud uvažujeme spontánní řeč (tj. nečtenou, zatíženou neřečovými událostmi jako je hlasité váhání apod.) mluvenou velkým množstvím různých řečníků (různé tempo a barva hlasu, různé přízvuky, atd.), využívajících slovník o několika desetitisících až milionech slov, kteří mluví v různých prostředích s různou mírou šumů a zvuků v pozadí. Z důvodu výše zmíněné rozmanitosti je nezbytné, aby byly rozpoznávače řeči založeny na statistických metodách.

Předpokládejme, že $W = w_1, w_2, ..., w_N$ je posloupnost N slov, a nechť $O = o_1, o_2, ..., w_T$ je akustická informace, tj. posloupnost vektorů příznaků (viz 2.2), odvozená z řečového sig-

nálu, z níž se snažíme rozpoznat, která slova byla vyslovena. Cílem je nalézt posloupnost slov $\tilde{W}$, která maximalizuje podmíněnou pravděpodobnost $P(W|O)$, jinak řečeno, hledáme nejpravděpodobnější posloupnost slov $\tilde{W}$ pro danou akustickou informaci O. Proces hledání posloupnosti $\tilde{W}$ se nazývá **dekódování** a bude podrobněji popsán v podkapitole 2.7.

Použitím Bayesova pravidla získáme rovnici 2.2.

$$\tilde{W} = \arg\max_{W} P(W|O) = \arg\max_{W} \frac{P(W)P(O|W)}{P(O)} \tag{2.2}$$

Kde $P(O|W)$ je pravděpodobnost, že při vyslovení posloupnosti slov W bude produkována posloupnost výstupních vektorů příznaků O, $P(W)$ je apriorní pravděpodobnost posloupnosti slov W a $P(O)$ je apriorní pravděpodobnost posloupnosti výstupních vektorů. Protože pravděpodobnost $P(O)$ není funkcí W, lze ji při hledání maxima v rovnici 2.2 ignorovat. Hledanou pravděpodobnost $\tilde{W}$ lze tedy nalézt maximalizací sdružené pravděpodobnosti $P(W, O)$, viz rovnice 2.3.

$$\tilde{W} = \arg\max_{W} P(W, O) = \arg\max_{W} P(W)P(O|W) \tag{2.3}$$

Z rovnice 2.3 vyplývá, že problém nalezení nejlepší posloupnosti slov k danému akustickému signálu je závislý na dvou oddělených pravděpodobnostech $P(W)$ a $P(O|W)$, které mohou být trénovány a modelovány nezávisle na sobě.

Podmíněné rozdělení pravděpodobnosti $P(O|W)$ nese informaci o **akustickém modelu** (viz 2.4) a apriorní rozdělení pravděpodobnosti $P(W)$ nese informaci o **jazykovém modelu** (viz 2.5). Tyto dva zdroje pravděpodobností je nutné určit ještě před vlastním rozpoznáváním, obvykle formou trénování z řečových a jazykových (textových) dat. Informace uvedené v této podkapitole byly převzaty z [12, 8].

2.4 Akustický model

Úkolem akustického modelu je co nejrychleji a nejpřesněji získat odhad podmíněné pravděpodobnosti $P(O|W)$ (popsané v podkapitole 2.3) pro libovolné O a W. V současnosti nejpoužívanější jsou pro akustické modelování skryté Markovovy modely (anglicky *Hidden Markov Model*, značeno *HMM*).

Princip metody modelování řeči Markovovými modely vychází z představy o vytváření řeči. Při generování řeči člověkem si lze představit, že hlasové ústrojí je během krátkého časového intervalu (např. rámce) v jednom z konečného počtu stavů artikulačních konfigurací. V uvažovaném mikrosegmentu je pak hlasovým ústrojím generován krátký signál závislý na stavu artikulačního ústrojí, a může být popsán určitými spektrálními charakteristikami, jež jsou reprezentovány vhodnými příznaky (viz 2.2).

Úloha nalezení co nejlepšího odhadu rozdělení pravděpodobnosti $P(O|W)$ je v našem případě určení topologie skrytého Markovova modelu a hodnot jeho parametrů.

Skrytý Markovův model je model stochastického procesu, na něhož lze pohlížet jako na pravděpodobnostní konečný automat, který v diskrétních časových okamžicích generuje posloupnost vektorů pozorování $O = o_1 o_2 ... o_T$. V každém časovém kroku změní model svůj stav s_i podle souboru předem daných pravděpodobností přechodu a_{ij}. Stav s_j, do kterého model přejde, generuje příznakový vektor o_t, a to podle rozdělení výstupní pravděpodobnosti $b_j(o_t)$ příslušné k tomuto stavu.

Pravděpodobnosti přechodu a_{ij} jsou podmíněné a určují, s jakou pravděpodobností přechází model ze stavu s_i, v kterémkoli čase t, do stavu s_j v čase $t+1$. Platí tedy:

$$a_{ij} = P\big(s(t+1) = s_j \big| s(t) = s_i\big) \tag{2.4}$$

kde $s(t)$ je stav modelu v čase t. Dále se předpokládá, že pravděpodobnosti přechodů a_{ij} jsou v čase konstantní, a že se sumují do jedničky, tedy:

$$\sum_{j=1}^{N} a_{ij} = 1 \tag{2.5}$$

Funkce rozdělení pravděpodobnosti $b_j(o_t)$ popisují rozdělení pravděpodobnosti pozorování o_t produkovaného ve stavu s_j v čase t a platí pro ně:

$$b_j(o_t) = P(o_t|s(t) = s_j) \tag{2.6}$$

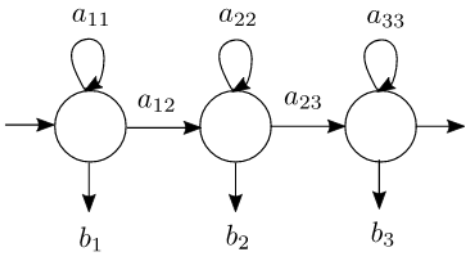
a současně:

$$\sum_{o} b_j(o) = 1 \tag{2.7}$$

pro všechny emitující stavy (tj. stavy, jež jsou schopny generovat výstupní vektor příznaků).

Při modelování řeči se používají zejména tzv. **levo-pravé Markovovy modely** (anglicky *left-to-right models*), které se používají pro modelování procesů, jejichž vývoj je spojen s postupujícím časem. S postupujícím časem dochází k přechodům ze stavů s nižšími indexy do stavů s vyššími indexy (tj. zleva doprava). [12]

Při práci s rozměrnějšími slovníky (tisíce až desetitisíce slov) a potřebě alespoň několika trénovacích promluv pro každé slovo vyvstává u modelů celých slov problém s těžkopádným a dlouho trvajícím trénováním klasifikátoru. Proto se v současnosti pro zjednodušení procesu trénování parametrů skrytých Markovových modelů slov odvozují od menších jednotek řeči – fonémů. Velice často používaný levo-pravý Markovův model jednoho fonému je znázorněn na obrázku 2.2.



Obrázek 2.2: Levo-pravý Markovův model fonému s 3 emitujícími stavy. [18]

Výstupem akustického modelu je poté pro každý rámec promluvy vektor pravděpodobností, kde jednotlivé hodnoty výstupního vektoru odpovídají jednotlivým pravděpodobnostem výskytu konkrétních fonémů daného jazyka.

Dříve bylo pro modelování rozložení pravděpodobností jednotlivých fonémů používáno GMM (Gaussian mixture model), v současnosti jsou však více osvědčené akustické modely postavené na hlubokých neuronových sítích (anglicky *Deep Neural Network*, zkr. *DNN*) viz např. [6].

2.5 Jazykový model

Každý jazyk používá slovník obsahující výčet slov (a jejich výslovnosti v podobě sekvencí fonémů) a pravidla pro řetězení těchto slov. Získáním informací o slovníku a zákonitostech řetězení slov se zabývá jazykové modelování. Úkolem jazykového modelu je poskytnout dekodéru co nejrychleji a nejpřesněji odhad apriorní pravděpodobnosti $P(W)$ (viz 2.3) pro libovolnou posloupnost slov W. Takto definovaný jazykový model se nazývá **stochastický jazykový model**.

Úkolem stochastického jazykového modelu je stanovit pro každou posloupnost W apriorní pravděpodobnost $P(W)$ této posloupnosti. Pravděpodobnost posloupnosti W, čítající K slov, je možné určit podle vztahu z rovnice 2.8.

$$\begin{aligned} P(W) &= P(W_1^K) = P(w_1 w_2 .. w_K) \\ &= P(w_1)P(w_2|w_1)P(w_3|w_1 w_2)...P(w_K|w_1 w_2 .. w_{K-1}) \\ &= P(w_1)P(w_2|w_1^1)P(w_3|w_1^2)...P(w_K|w_1^{K-1}) \\ &= \prod_{i=1}^{K} P(w_i|w_1^{i-1}) \end{aligned} \tag{2.8}$$

V uvedeném rozkladu pravděpodobnosti $P(W)$ v rovnici 2.8 jsou podmíněné pravděpodobnosti výskytu slova w_i podmíněny pouze svojí historií, tj. posloupností $w_1...w_{i-1}$. Tento fakt nám umožňuje provádět proces dekódování již během promlouvání, nikoli až po skončení celé rozpoznávané promluvy.

Přesný výpočet apriorních pravděpodobností $P(W_1^K)$ všech posloupností slov W libovolné délky K není téměř možné získat. Provadí se proto jejich aproximace, kdy všechny historie $w_1...w_{i-1}$, které se shodují v posledních n slovech, jsou zařazeny do stejné třídy. Tyto stochastické modely jsou tzv. **n-gramové modely**. N-gramem se rozumí posloupnost n za sebou jdoucích slov z náhodné pozice, např. v trénovacím korpusu. Pro $n = 1$ mluvíme o unigramech, $n = 2$ bigramech a $n = 3$ trigramech. Rozpoznávače řeči používající statický dekodér obvykle nevyužívají n-gramy vyššího řádu z důvodu překročení limitů potřebné operační paměti, u dynamického dekódování tomu tak být nemusí.

V n-gramovém jazykovém modelu je pravděpodobnost $P(w_1^K)$ aproximována na základě posledních $n-1$ slovech historie, viz rovnice 2.9.

$$P(w_1^K) \approx \prod_{i=1}^{k} P(w_i|w_{i-n+1}^{i-1}) \tag{2.9}$$

Odhad n-gramové pravděpodobnosti $\overline{P}(w_k|w_{k-2}w_{k-1})$ pro nás neznámé pravděpodobnosti $P(w_k|w_{k-2}w_{k-1})$ získáme z rovnice 2.10:

$$\overline{P}(w_k|w_{k-2}w_{k-1}) = \frac{N(w_{k-2}, w_{k-1}, w_k)}{N(w_{k-2}, w_{k-1})} \tag{2.10}$$

Kde $N(w_{k-2}, w_{k-1}, w_k)$ je počet, kolikrát se trigram $w_{k-2}w_{k-1}w_k$ objevil v trénovacích datech (rozsáhlém korpusu daného jazyka) a $N(w_{k-2}, w_{k-1})$ je počet, kolikrát se v korpusu vyskytl bigram $w_{k-2}w_{k-1}$.

N-gramové modely jsou velmi účinné pro jazyky s poměrně pevným pořadím slov ve větě a s co nejmenším počtem slov. Jako příklad uveďme angličtinu a češtinu. Čeština kvůli ohebnosti slov disponuje výrazně většími slovníky, a také nemá tak pevný pořádek slov ve větě.

Z tohoto důvodu vytvořit n-gramový model pro češtinu představuje značně komplikovanější úlohu (a obecně lze předpokládat, že výsledná kvalita rozpoznávání nebude tak vysoká).

Hlavním problémem n-gramových modelů je nedostatek trénovacích dat. Uvažujme jazyk se slovníkem čítající 50 000 slov, potom existuje $50\,000^3 = 1,25 * 10^{14}$ potenciálních trigramů. Takové množství dat není prakticky možné pořídit. S problémem nedostatku trénovacích dat se dobře vypořádají například jazykové modely založené na rekurentních neuronových sítích (viz např. [9]).

Z nedostatku trénovacích dat může dojít k odhadu nulové pravděpodobnosti n-gramu. Skutečnost, že se slovní spojení nenachází v trénovacím korpusu neznamená, že toto slovní spojení není možné, proto bývají používány metody vyhlazování (anglicky *smoothing*). Jednou z těchto metod je tzv. **ústupové schéma** (*back-off schema*), které spočívá v přerozdělení pravděpodobností tak, aby k odhadu nulových pravděpodobností nedocházelo.

Vyhlazený odhad těchto pravděpodobností se počítá pomocí rozdělení s ústupovou (*backing-off*) historií $\beta(w|\overline{h})$. Výpočet $\beta(w|\overline{h})$ se typicky provádí pomocí relativní četnosti $(n-1)$-gramů viz rovnice 2.11.

$$\beta(w|\overline{h}) = \frac{N(\overline{h}, w)}{N(\overline{h})} \tag{2.11}$$

Kde

$N(\overline{h}, w)$ je počet výskytů posloupnosti $\overline{h}w$ v trénovacím korpuse,

$N(\overline{h})$ je počet výskytů konktértní historie $\overline{h}$,

$\overline{h}$ je **zobecněná historie** - tj. typicky historie h zkrácená o nejstarší slovo.

Pravděpodobnosti pozorovaných n-gramů jsou zde odhadovány na základě jejich relativních četností $\frac{N(h,w)}{N(h)}$, které jsou vždy sníženy součinitelem $d_{N(h,w)}$. Tímto způsobem ubraná část pravděpodobnosti se pak přerozdělí všem nepozorovaným jevům, tj. n-gramům, jejichž počet $N(h, w) = 0$. Číslo $0 \leq d_{N(h,w)} \leq 1$ se nazývá **diskontní činitel** (*discount factor*), neboť způsobuje, že odhad pravděpodobností n-gramů je snížen v poměru $d_{N(h,w)}$. Odhady nepozorovaných jevů pak získáme vynásobením pravděpodobností zobecněného rozdělení ústupovou váhou $B(h)$, která zahrnuje jak přerozdělené množství, tak i normalizační člen. Formálně lze ústupové schéma charakterizovat následovně

$$P_{BO}(w|h) = \begin{cases} d_{N(h,w)} \frac{N(h,w)}{N(h)} & \text{pro } N(h,w) > 0 \\ B(h)\beta(w|\overline{h}) & \text{pro } N(h,w) = 0 \end{cases} \tag{2.12}$$

Kde $B(h)$ vypočteme vztahem z následující rovnice 2.13

$$B(h) = \frac{1 - \sum\limits_{w:N(h,w)>0} d_{N(h,w)} \frac{N(h,w)}{N(h)}}{\sum\limits_{w:N(h,w)=0} \beta(w|\overline{h})} \tag{2.13}$$

Text této podkapitoly byl inspirován knihou [12] a lze zde získat podrobnější informace o jazykovém modelování včetně různých metod tvorby n-gramového jazykového modelu či způsobech posuzování kvality stochastických jazykových modelů.

2.6 Rozpoznávací síť

Na rozpoznávací síť lze pohlížet jako na ohodnocený orientovaný graf, který může obsahovat smyčky, a jehož každý uzel představuje jedno slovo (může jít i o prázdné slovo) či stav pravděpodobnostního konečného automatu akustického či jazykového modelu, jehož orientovaná ohodnocená hrana reprezentuje možný přechod z daného slova na další slovo či z daného stavu na další stav. [12] Obvykle se pro ohodnocení hrany používá logaritmus pravděpodobnosti přechodu, s cílem eliminace problému násobení počítačově reprezentovaných reálných čísel (docházelo by ke ztrátě přesnosti). Na úrovni logaritmů je násobení nahrazeno sčítáním, viz rovnice 2.14.

$$\log(x_1 * x_2 * ... * x_N) = \log(x_1) + \log(x_2) + ... + \log(x_N) \tag{2.14}$$

Velmi časté je aplikování tzv. **konečných převodníků**, jejichž myšlenka spočívá ve snaze vytvořit celou rozpoznávací síť před vlastním procesem dekódování, a tuto síť optimalizovat jednak snížením počtu redundantních částí cest v síti ve smyslu Viterbiho kritéria (tj. ze všech částí cest grafu reprezentujících stejnou posloupnost symbolů – stavů akustického modelu a přechodů – stačí uvažovat jen jednu nejlepší, viz 2.7.2), a jednak spojením co nejvíce shodných částí cest. Při tvorbě sítě se použijí všechny dostupné zdroje znalostí o dané úloze, a takto vzniklá komplexní rozpoznávací síť se optimalizuje. Optimalizace sítě, tedy minimalizace počtu stavů a hran, je pak založena na použití operace determinizace (anglicky *determinization*), stlačení vah (anglicky *weight pushing*) a minimalizace (anglicky *minimalization*). Výsledná optimalizovaná síť je pak v průběhu dekódování prohledávána algoritmem Token Passing (využívající Viterbiho kritéria). [12]

Kvůli složitému procesu vytváření rozpoznávací sítě, jež vyžaduje optimalizování pomocí operací determinizace, stlačení vah a minimalizace, nelze dopředu předvídat její výslednou velikost. Je tedy i komplikované posoudit, jak velký jazykový model můžeme do rozpoznávací sítě zakomponovat, aby následné dekódování netrpělo problémem nedostatku operační paměti. Tento problém velmi dobře odstraňuje dynamický dekodér, který pracuje pouze s malou rozpoznávací sítí, a pro který jsme schopni předem přesně spočítat, jaké množství paměti bude jazykový model v operační paměti zabírat.

2.7 Dekodér a dekódování řeči

V podkapitole 2.3 bylo definováno dekódování jako proces hledání posloupnosti slov $\tilde{W}$, jež maximalizuje součin pravděpodobností $P(W)$ a $P(O|W)$. Přičemž $P(W)$ a $P(O|W)$ jsou funkce, u kterých jsme schopni spočítat jejich hodnotu pro libovolnou posloupnost slov W, a které jsou získané separátně z jazykového a akustického modelu (viz 2.4 a 2.5).

Cíl dekódování pak popisuje rovnice 2.15.

$$\tilde{W} = \arg\max_{W} P(W)P(O|W) \tag{2.15}$$

Někdy je úloha dekódování zobecněna na hledání N nejlepších (tj. nejpravděpodobnějších) posloupností slov $\tilde{W}$ (anglicky *N-best*). [12]

2.7.1 Výpočetní problém dekódování

Dekódovací algoritmus, jenž by provedl úplné prohledání všech posloupností slov W v úloze se slovníkem o V slovech a průměrné délce posloupnosti slov M, by musel vyčíslit až V^M

posloupností slov, tj. 10^{50} vyhodnocení v případě $V = 100\,000$ a $M = 10$. Pro každou takovou posloupnost by dále musel spočítat hodnotu součinu $P(W)P(O|W)$.

Pro výpočet $P(O|W)$ je pak nutné určit všechny stavové posloupnosti a pro každou tuto stavovou posloupnost dále spočítat, s jakou pravděpodobností akustický model produkuje uvažovanou posloupnost pozorování O. Každá stavová posloupnost má délku rovnu délce posloupnosti pozorování O. Označíme-li tuto délku T, pak pronese-li řečník v průměru 10 slov za 5 sekund, a uvažujeme-li velikost jednoho rámce $10\,ms$, bude $T \approx 500$. Má-li každý stav třeba jen dva možné následníky, je možné vytvořit až 2^{500} různých stavových posloupností pro každou z 10^{50} posloupností slov.

Z výše uvedeného však plyne, že pro 5 sekund řeči by bylo nutné provést $2^{500} * 10^{50} * 500 \approx 10^{203}$ zpracování stavu. Takováto strategie dekódování však není realizovatelná. Systémy rozpoznávání řeči musí posloupnost $\tilde{W}$ nalézt obvykle i v reálném čase, proto hlavním problémem procesu dekódování není pouze nalezení této posloupnosti $\tilde{W}$, ale jak tuto posloupnost nalézt v reálném čase. [12]

2.7.2 MAP a Viterbiho kritérium

Z podkapitoly 2.4 víme, že akustické modely jsou v dnešní době takřka vždy budovány pomocí skrytých Markovových modelů. Symbolem S označme stavovou posloupnost, kterou může takový akustický model projít při generování posloupnosti pozorování O. Akustický model $P(O|W)$ je pak možné vyjádřit zápisem:

$$P(O|W) = \sum_{S} P(O|S,W)P(S|W) \tag{2.16}$$

kde $P(O|S,W)$ je podmíněná pravděpodobnost jevu, že model vygeneruje posloupnost příznaků O při průchodu posloupnosti slov W a posloupnosti stavů S, a $P(S|W)$ je podmíněná pravděpodobnost, že model projde posloupností stavů S patřící posloupnosti slov W. Protože pravděpodobnost generování posloupnosti pozorování O konečným pravděpodobnostním automatem je u skrytých Markovových modelů závislá pouze na stavové posloupnosti S a na parametrech přiřazených každému stavu $s \in S$ stavové posloupnosti, kterou model projde. Platí:

$$P(O|S,W) = P(O|S) \tag{2.17}$$

rovnici 2.16 lze tedy přepsat následujícím způsobem:

$$P(O|W) = \sum_{S} P(O|S)P(S|W) \tag{2.18}$$

Dosazením vztahu z 2.18 do rovnice 2.15 získáme následující formulaci úlohy dekódování:

$$\tilde{W} = \arg\max_{W} \; P(W) \sum_{S} P(O|S)P(S|W) \tag{2.19}$$

kde S je stavová posloupnost, kterou může model s konečným počtem stavů projít při generování posloupnosti pozorování O.

Označme symbolem Φ množinu všech stavových posloupností S všech možných posloupností slov a Φ_W množinu všech stavových posloupností S reprezentujících konkrétní posloupnost slov W (platí $\Phi_W \subset \Phi$). Pravděpodobnost $P(S|W)$ je nenulová jen pro $S \in \Phi_W$, nemá tedy smysl ve vztahu 2.19 sčítat přes všechna S, ale jen přes ta, pro která není pravděpodobnost $P(S|W)$ nulová. Označíme-li S_W posloupnost stavů modelu reprezentujícího

pouze posloupnost slov W a definujeme-li $P(S_W) = P(S|W)$, je pro nenulové sčítance v 2.19 $P(O|S) = P(O|S_W)$ a rovnici 2.19 lze přepsat na tvar:

$$\begin{aligned} \tilde{W} &= \arg\max_W \; P(W) \sum_{S \in \Phi_W} P(O|S)P(S|W) \\ &= \arg\max_W \; P(W) \sum_{S_W \in \Phi_W} P(O|S_W)P(S_W) \end{aligned} \tag{2.20}$$

Vztah 2.20 nám definuje úlohu dekódování podle **kritéria MAP** (*maximum a posteriori probability*).

Experimentálně bylo zjištěno, že uvážíme-li v součtu pravé strany výrazu 2.19 pouze nejpravděpodobnější stavovou posloupnost S, přesnost dekódování se v podstatě nezmění. Po uplatnění této aproximace dostáváme tzv. **Viterbiho kritérium dekódování**, viz rovnice 2.21.

$$\begin{aligned} \tilde{W} &= \arg\max_W \; P(W) \max_S P(O|S)P(S|W) \\ &= \arg\max_W \; P(W) \max_{S_W \in \Phi_W} P(O|S_W)P(S_W) \end{aligned} \tag{2.21}$$

Viterbiho kritérium v podstatě převádí problém dekódování na problém hledání nejpravděpodobnější cesty, tj. cesty s nejmenší cenou v ohodnoceném orientovaném grafu (pro hrany ohodnocené zápornými logaritmy pravděpodobnosti) tvořeném sítí s konečným počtem stavů akustického modelu. Nad problémem dekódování lze tedy uvažovat jako nad úlohou prohledávání stavového prostoru. Jelikož je však tento stavový prostor v případě dekódování s velkými slovníky obrovský (viz 2.7.1) jsou pro řešení používány různé suboptimální strategie (nezaručují nalezení nejpravděpodobnější posloupnosti slov $\tilde{W}$), které si kladou za cíl dosáhnout co nejlepšího kompromisu mezi rychlostí a přesností řešení. [12]

2.7.3 Algoritmus Token Passing

Algoritmus **Token Passing** je konkrétní implementací Viterbiho algoritmu hledání minimální cesty v orientovaném grafu. Není nutné si pamatovat všechny minimální přechody ze stavů do stavu, nýbrž pouze minimální přechody mezi slovy.

Algoritmus byl poprvé uveden v [17] a na jeho principu je implementován také dekodér open source projektu **Kaldi** (viz [11]), jehož modifikace je implementací statického dekodéru, jež je součástí **BSCORE** firmy **Phonexia** (viz [3]). Tato podkapitola využívá popisu algoritmu uvedeném v [12].

Definujme stavovou mřížku, jejíž každý uzel je určen uspořádanou dvojicí $[u, t]$, kde u je stavem HMM a t je časový krok. V časovém kroku t je v každém uzlu $[u, t]$ umístěn **token**, který se při každé změně časového kroku z t na $t+1$ přemístí do všech přímých následníků uzlu u, a který nese informaci o posloupnosti slov (včetně její ceny), jíž prošel od počátku dekódování. Setká-li se v čase t a uzlu u více tokenů, je z nich ponechán jen ten, jenž prošel posloupností slov prokazujících nejmenší cenu. Na začátku dekódování v čase $t = 1$ existuje v počátečním neemitujícím stavu jen jeden token s nulovou cenou, všechny ostatní stavy obsahují tokeny s nekonečně velkou cenou. Uskuteční-li token mezislovní přechod, aktualizuje si informaci o posloupnosti slov, kterou prošel (tj. rozšíří si doposud zapamatovanou informaci o právě prošlé slovo).

Formálněji lze algoritmus popsat následujícím způsobem. Každý token je uspořádaná dvojice $[\Phi_j(t), \Psi_j(t)]$, kde $\Psi_j(t)$ značí identifikátor cesty, kterou token uložený ve stavu s_j

prošel od počátku do času t, a kde $\Phi_j(t)$ je cena této cesty. Identifikátor cesty je realizován jako zřetězený seznam, jehož položky jsou značeny zkratkou **WLR** (z anglického *word link record*). Každý WLR záznam obsahuje identifikátor slova (obvykle index ze slovníku) a odkaz na předchozí WLR (tj. tvoří strukturu zřetězeného seznamu). Pro každý token $[\Phi_j(t), \Psi_j(t)]$, který při mezislovním přechodu při změně časového kroku t na $t+1$ opustí konec slova w, se vytvoří nový WLR obsahující identifikátor prošlého slova w a identifikátor cesty $\Psi_j(t)$. Záznam WLR může též obsahovat položku $\Phi_j(t)$ a informaci o času mezislovního přechodu $t+1$ (jestliže jako součást výstupu dekódování chceme informaci o časových hranicích mezi slovy a jejich pravděpodobnost). Identifikátor cesty tokenu v koncovém stavu a na konci dekódování (tj. v čase T) nám pak zpětným zřetězením určuje hledanou posloupnost slov s minimální cenou.

Popis algoritmu Token Passing

1. **Inicializace v čase** $t = 1$

$$\Phi_j(1) = \begin{cases} -\log b_j(o_1) - \log a_{1j} & \text{pro } j = 2, ..., N-1 \\ \infty & \text{pro } j = 1 \end{cases}$$
$$\Psi_j(1) = 0 \qquad \text{pro } j = 1, .., N-1$$

2. **Pro všechny časy** $t = 2, ..., T$ **proveď**

- Pro všechny stavy s_i, kde $i = 2, ..., N-1$ zkopíruj při změně časového kroku z $t-1$ na t token ze stavu s_i do všech stavů s_j, do kterých existuje přechod a zvětši cenu každého zkopírovaného tokenu:

  for $i = 2, ..., N-1$ **do**
    for $j = 2, ..., N-1$ **do**
      $\Phi_{ij}(t) = (\Phi_i(t-1) - \log a_{ij}) - \log b_j(o_t)$
      $\Psi_{ij}(t) = \Psi_i(t-1)$
    end for
  end for

  kde $\Phi_{ij}(t)$ je nová cena tokenu zkopírovaného ze stavu s_i do s_j v čase t a $\Psi_{ij}(t)$ je identifikátor této cesty.

- Pro všechny stavy s_j, kde $j = 2, ..., N-1$ najdi ze všech tokenů zkopírovaných v čase t do stavu s_j token s nejlepším ohodnocením cesty $\Phi_{mj}(t)$ a všechny ostatní tokeny tohoto stavu odstraň. Pokud nalezený token při přechodu z času $t-1$ do času t vykonal mezislovní přechod, vytvoř pro něj nový záznam WLR:

  for $j = 2, ..., N$ **do**
    $m = \underset{i=2}{\overset{N-1}{\arg\min}} \ \Phi_{ij}(t)$
    $\Phi_j(t) = \Phi_{mj}(t)$
    $\Psi_j(t) = \Psi_{mj}(t)$
    if $IsCrossWord(m, j)$ **then**
      $\Psi_j(t) = CreateWLR(\Phi_m(t-1), \Psi_m(t-1), t, w(m))$
    end if
    for $i = 2, ..., N-1$ **do**
      delete $\Phi_{ij}(t)$
      delete $\Psi_{ij}(t)$

end for
end for

kde $IsCrossWord(x, y)$ je splněno, pokud přechod ze stavu s_x do s_y je přechodem mezislovním, $w(x)$ je funkce vracející identifikátor slova s koncovým stavem s_x a funkce $CreateWLR(\Phi, \Psi, t, w)$ je funkce, jenž vrací identifikátor cesty rozšířený o nové slovo w v čase t.

- Odstraň staré tokeny z času $t-1$

  for $i = 2, ..., N-1$ **do**
  delete $\Phi_i(t-1)$
  delete $\Psi_i(t-1)$
  end for

3. **Výsledek**

   - Token, jenž přijde do koncového neemitujícího stavu s_N v čase T s nejmenší hodnotou cesty udává index minimální cesty (tj. hledanou posloupnost slov) $\Psi_N(T^+)$ a její minimální cenu $\Phi_N(T^+)$:

     $$m = \arg\min_{i=2}^{N-1} \; (\Phi_i(T) - \log a_{iN})$$
     $$\Phi_N(T^+) = \Phi_m(T) - \log a_{mN}$$
     $$\Psi_N(T^+) = CreateWLR(\Phi_m(T), \Psi_m(T), T, w(m))$$

   - Nejpravděpodobnější posloupnost slov pak získáme zpětným trasováním $\Psi_N(T^+)$.

Pro účely zefektivnění dekódování algoritmem Token Passing se u dekodéru běžně používají dvě optimalizace: **beam pruning** a **live states pruning**. Prořezáváním (anglicky *pruning*) je zde myšlen proces eliminace nadbytečných tokenů. Omezení počtu aktivních tokenů tak vede k lepšímu časovému a paměťovému chování dekodéru.

Parametr **beam pruning** udává šířku intervalu definovaného jako $\langle best + beam; best \rangle$, kde best je cena nejlepšího tokenu a *beam* je šířka tohoto intervalu. Pokud dojde ke změně ceny mimo tento interval, je token zahozen a dále není uvažován.

Parametr **live states pruning** udává maximální počet současně existujících živých stavů (tj. stavů, ve kterých je alespoň jeden aktivní token). Pokud je počet živých stavů překročen, jsou tokeny nejhorších nadbytečných stavů smazány.

Hodnoty parametrů **beam pruning** a **live states pruning** jsou pro statický dekodér firmy Phonexia vhodně zvoleny na základě experimentů. Optimálním nastavením se ukázalo být $beam = 190$ a $max\ live\ states = 8000$.

2.7.4 Dynamický dekodér

V případě výše uvedeného popisu dekodéru s n-gramovou rozpoznávací sítí se jedná o statický dekodér. Princip statického dekódování spočívá ve vytvoření rozpoznávací sítě zkonstruované nad n-gramovým (obvykle využitím unigramů, bigramů a trigramů) jazykovým modelem. Tato rozpoznávací síť je načtena do operační paměti před samotným dekódováním a proces dekódování poté znamená pouze aplikování algoritmu Token Passing (z podkapitoly 2.7.3) nad grafem, jenž je dán uloženou rozpoznávací sítí.

Hlavní myšlenkou dynamického dekodéru je vyhnutí se vytváření celé rozpoznávací sítě před procesem dekódování, přičemž síť je generována (simulována) až při samotném dekódování. V případě, že je dynamicky generována rozpoznávací síť pouze ze základních komponent – tj. z akustických jednotek, výslovnostního slovníku a jazykového modelu – mluvíme

o **plně dynamickém dekódování**. Dynamického dekódování lze však také dosáhnout například nahrazením n-gramové rozpoznávací sítě unigramovou sítí, přičemž jazykový model je načten do operační paměti v podobě vhodně zvolené vyhledávací struktury, která slouží pro poskytování n-gramových pravděpodobností v průběhu dekódování (tj. dynamicky je generován pouze jazykový model). Pro dekódování pak použijeme modifikaci algoritmu token passing (viz 2.7.3), která umožňuje v případě, kdy se při přechodu ze stavu s_i do stavu s_j, ve stavu s_j setká více tokenů, neponechat pouze token s nejlepším ohodnocením cesty, nýbrž N (např. $N = 10$) tokenů s nejlepším ohodnocením. V případě, že dojde při přechodu z času t do času $t + 1$ k mezislovnímu přechodu, je upravena cena tokenů, jenž byla dána unigramovou rozpoznávací sítí pomocí pravděpodobnosti získané z vyhledávací struktury uchovávající n-gramové pravděpodobnosti. Pokud tedy dojde k mezislovnímu přechodu, je ze struktury WLR, příslušející danému tokenu, získána posloupnost rozpoznávaných slov, a pro tuto posloupnost je v n-gramové vyhledávací struktuře nalezen n-gram nejvyššího dostupného řádu. Na základě jeho pravděpodobnosti je modifikována doposud určená cena cesty. V případě, že není pro danou posloupnost slov nalezen n-gram vyššího řádu, použije se back-off pravděpodobnost (u unigramu se ponechá pravděpodobnost daná rozpoznávací sítí).

2.8 Následné zpracování

Následné zpracování (anglicky *postprocessing*) je závěrečnou fází rozpoznávání řeči. Jedná se o transformaci surového přepisu řeči, který je výstupem z dekodéru (viz 2.7), do čitelnější podoby. Surový přepis často sestává pouze z čistého textu bez rozlišování velkých a malých písmen, bez interpunkce (čárky, tečky, apod.), číslovky mohou být přepsány pouze slovy, atd. Jednoduchého následného zpracování lze dosáhnout i pouhou sadou regulárních výrazů nahrazujících jednotlivé textové elementy přepsané řeči. Tento přístup může výstup dekodéru znatelně zčitelnit, avšak nedosahuje takových výsledků, jako některé sofistikovanější metody (viz např. [14, 16]).

Kapitola 3

Hodnocení úspěšnosti rozpoznávání

Při implementaci rozpoznávače řeči je velmi důležité disponovat mechanismem, jenž dokáže objektivně posoudit, jak dobře daný rozpoznávač funguje. Hodnotící mechanismus tedy poskytuje obvykle číselné ohodnocení kvality přepisu řeči, přičemž výsledek je pro stejná přepisovaná data a stejný rozpoznávač vždy totožný. Hodnocení nějakým takovýmto mechanismem nazýváme **skórování** a hodnotu výsledku nazýváme **skóre**.

V této práci je skórování inspirováno NIST (*National Institute of Standards and Technology*) standardem RT09 (*Rich Transcription 2009*) vydávaného ministerstvem obchodu USA (*U.S. Department of Commerce*), viz [1, 4].

3.1 Skórování

Dle standardu [4] by měly být výše popsanou skórovací metodou vyhodnocovány pouze anglické promluvy (což nám při tvorbě dynamického dekodéru pro anglický jazyk nevadí), obsahující pouze alfabetické znaky, apostrofy, pomlčky a mezery. Slova by měla být porovnávána bez ohledu na malá a velká písmena. Hláskovaná slova by měla být uvedena pomocí písmen oddělených mezerami (například hláskování jména Smith bude zapsáno jako `s m i t h`). Skórovány by také měly být pouze lexikální části promluvy. Části obsahující kašlání, kýchání, hlasité dýchání, smích a jiné nelexikální části by neměly být do skórování zahrnuty.

U výše zmíněného standardu je skórem hodnota **WER** (z anglického *Word Error Rate*), jež je dána vztahem

$$WER = \frac{N_{Del} + N_{Ins} + N_{Subst}}{N_{Ref}} \tag{3.1}$$

kde

N_{Del} = počet slov referenční transkripce nenamapovaných na výstup rozpoznávače.

N_{Ins} = počet slov výstupu rozpoznávače nenamapovaných na referenční transkripci.

N_{Subst} = počet slov výstupu rozpoznávače namapovaných na rozdílné slovo referenční transkripce.

N_{Ref} = počet slov referenční transkripce.

Čitatel rovnice 3.1 tedy tvoří tři typy chyb nazývané **deletions** (N_{Del}), **insertions** (N_{Ins}) a **substitutions** (N_{Subst}).

Jako skóre se také často používá hodnota **WA** (z anglického *Word Accuracy*), pro kterou platí:

$$WA = 1 - WER \tag{3.2}$$

Podotkněme, že hodnota WER se často uvádí v procentech a je potenciálně přípustnou hodnotou i $WER > 100\,\%$, a tedy i $WA < 0$. Avšak z praktického hlediska uvažovat takto nefunkční rozpoznávač postrádá smysl.

Abychom mohli spočítat hodnotu WER, je nutné provést tzv. zarovnání (anglicky *alignment*), jehož úkolem je zarovnat „pod sebe“ slova z výstupu rozpoznávače a z referenční transkripce tak, aby se minimalizovalo skóre WER. Pro NIST standard z [4] je použito zarovnání nástrojem ASCLITE popsaným v [5].

Na obrázku 3.1 lze vidět příklad zarovnání slov z referenční transkripce s výstupem z rozpoznávače. Velkým písmem jsou označena slova, u nichž došlo k chybě (I - *insertion*, D - *deletion*, S - *substitution*). Skóre takového příkladu lze vypočítat podle vzorce 3.1 následovně

$$WER = \frac{1+1+2}{11} = \frac{4}{11} \doteq 36.36\,\% \tag{3.3}$$

Reference:	good	morning	*	i	am	REALY	happy	that	WE	SET	this	appointment
Hypotéza:	good	morning	A	i	am	REALLY	happy	that	**	RESET	this	appointment
Evaluace:			**I**			**S**			**D**	**S**		

Obrázek 3.1: Příklad zarovnání slov s účelem minimalizovat hodnotu WER pro evaluaci.

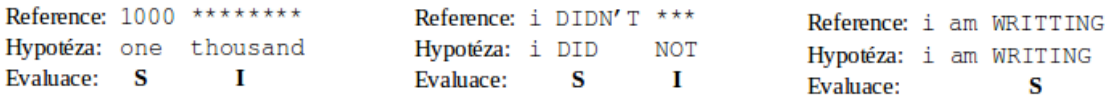
3.2 Datové sady

Pro správné skórování je nutné disponovat také datovými sadami (tzv. datasety), jež sestávají z různých nahrávek, které obsahují řeč v rozpoznávaném jazyce mluvenou v ideálním případě zkoumanou cílovou skupinou lidí. Pokud tedy chceme vytvořit univerzální rozpoznávač, měla by skórovací data obsahovat rovnoměrné rozložení žen a mužů, nahrávky by měly pocházet z různých zařízení, měly by být zatíženy různou úrovní šumu v pozadí (tj. pocházet z různých prostředí), měly by se týkat různých témat, apod. Častějším požadavkem je však vytvářet rozpoznávač pro konkrétnější případ využití (například přepisování nahrávek z call center), v takovém případě postačíme s užší skupinou skórovacích nahrávek (například s částí nahrávek pořízených v call centrech, do kterých je systém dodáván). Poznamenejme taktéž, že podobná pravidla jako pro skórovací data platí také pro data trénovací (systém by měl být trénován na datech, pro která je systém vytvořen).

Nutnou podmínku pro objektivnost výsledků skórování je velikost datové sady. Pokud dataset není dostatečně rozsáhlý, mohou být výsledky značně zkreslené náhodou. Pokud tedy zvážíme dostatečně velkou datovou sadu a skutečnost, že rozpoznávače řeči často nezvládají pracovat ani v reálném čase, je nutné proces skórování provádět ve výpočetním cloudu. Pro účely skórování v této práci byl využit SGE (*Sun Grid Engine*) viz [2].

Značnou komplikací je nutnost vlastnit k daným skórovacím nahrávkám i jejich ručně vytvořený přepis. Získat kvalitně vytvořené anotace nahrávek je pracné, nákladné a zatížené

lidskou chybou. Anotátor nahrávce nemusí správně rozumět, může se přepsat, či může zvolit jiný zápis, než který vygeneruje rozpoznávač (viz obrázek 3.2).



Obrázek 3.2: Typické chyby způsobené špatnou anotací.

Pro naše účely byly v této práci využity čtyři skórovací datasety, poskytnuté firmou Phonexia. Dva z nich pochází z amerických call center, z důvodu zachování jejich anonymity označme tyto datové sady jako **CC1** a **CC2** (a toto označení používejme i ve zbytku práce). Dalším datasetem je sada různých nahrávek pořízená z různých zařízení (především však iPad), označme jej jako **RZ**. Poslední datovou sadou jsou nahrávky z NIST evaluací z roku 2001, jež označme zkratkou **N01**. V tabulce 3.1 jsou uvedeny součty délek nahrávek pro jednotlivé datové sady.

Dataset	Hodin
CC1	2 h 0 m
CC2	4 h 16 m
RZ	0 h 11 m
N01	6 h 9 m

Tabulka 3.1: Velikosti jednotlivých datových sad (součty délek nahrávek jednotlivých datasetů).

3.3 Porovnávání časové a paměťové složitosti

Tato podkapitola zavádí výchozí podmínky pro hodnocení časové a paměťové náročnosti. Pokud u popisu experimentu nebude uvedeno jinak, bylo využito následujících podmínek.

Uvedený čas a paměť odpovídají průměrné hodnotě výstupů tří spuštění zkoumaného rozpoznávače unixovým programem `time` nad nahrávkou o délce 30.76 s, obsahující na základě výstupu detektoru řeči 18.81 s mluvené řeči v americké angličtině.

Měření je často prováděno za účelem porovnání statického a dynamického dekodéru, či dvou různých konfigurací dynamického dekodéru. Podotkněme, že pro všechny konfigurace zkoumané v této práci roste časová i paměťová náročnost lineárně s délkou řečové části nahrávky. Proto není pro měření nutné používat různě dlouhé nahrávky.

Měření bylo provedeno na procesoru Intel® Core™ i5-3470 CPU @ 3.20GHz s 16 GB RAM pamětí.

Kapitola 4

Rozpoznávač v BSCORE

V kapitole 2 byly popsány jednotlivé komponenty z nichž sestává každý rozpoznávač řeči (viz schéma na obrázku 2.1). V této kapitole detailněji popíšeme řešení těchto komponent firmou Phonexia v rámci implementace jejich rozpoznávače řeči, který je součástí BSCORE (jádra řečových technologií vyvíjených firmou Phonexia). Zjednodušené schéma tohoto rozpoznávače lze nalézt na obrázku 4.1.

Ve fázi předzpracování dojde k načtení vstupního souboru v některém ze zvukových formátů (wav, raw, flac, opus), převzorkování na vzorkovací frekvenci 8000 Hz a konverzi na formát *lin16* (tj. vzorky jsou 16b celá čísla). Pokud nahrávka obsahuje více kanálů, pak je zpracováván každý kanál odděleně a nezávisle na ostatních. S předzpracováním úzce souvisí také analýza řečových a neřečových rámců detektorem řeči, a to ve dvou fázích. První fáze filtruje neřečové rámce na základě energetické analýzy signálu. Druhá fáze detekce neřečových rámců je založena na klasifikaci pomocí neuronové sítě, přičemž vstupem této neuronové sítě jsou některé z extrahovaných příznaků.

Extrakce příznaků (*feature extraction*) spočívá ve výpočtu MFCC příznaků (viz 2.2), odhadu základního hlasivkového tónu (značeno F0) algoritmem použitým v open source projektu Kaldi (viz [11]) a algoritmem, jenž byl vytvořen firmou Phonexia (BSCORE F0).

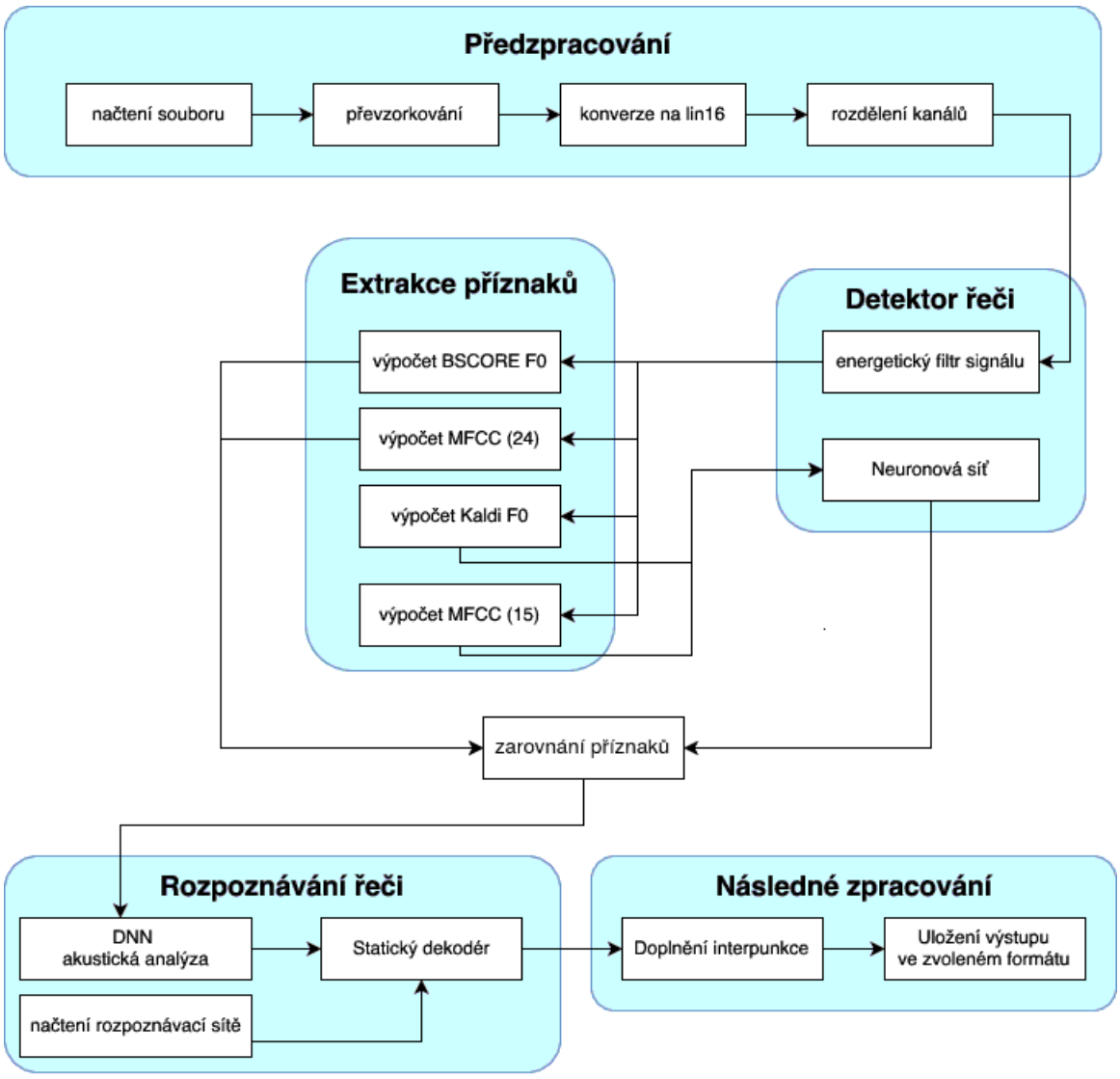
Extrahované příznaky řečových rámců vstupují do DNN (*deep neural network*), jenž vyhodnocuje pravděpodobnosti jednotlivých stavů fonémů akustického modelu (detailnější informace o tomto přístupu lze nalézt např. v [6]). Výstupem této neuronové sítě jsou tedy pravděpodobnosti jednotlivých stavů fonémů, které spolu s načtenou trigramovou rozpoznávací sítí vstupují do statického dekodéru, kde probíhá dekódování na výstupní sekvenci slov algoritmem Token Passing viz 2.7.3.

Výstupní sekvence slov z dekodéru je na závěr ve fázi následného zpracování doplněna o interpunkci (viz např. [14, 16]) a uložena do zvoleného formátu.

Dynamický dekodér, jenž je předmětem této práce, je implementován na místo statického dekodéru (napsaném v jazyce C++) jako součást rozpoznávače řeči popsaného v této kapitole. Proto také zde popsaný statický rozpoznávač definuje výchozí stav, tj. časovou a paměťovou náročnost a úspěšnost rozpoznávání, kterému by měl dynamický dekodér konkurovat. Tento výchozí stav je zachycen v tabulce 4.1.

Čas [s]	**Paměť** [KB]	**CC1** (WA)	**CC2** (WA)	**RZ** (WA)	**N01** (WA)
20.41	2 028 790	78.5 %	56.3 %	55.9 %	76.9 %

Tabulka 4.1: Referenční výsledky získané statickým dekodérem za podmínek z kapitoly 3.3.



Obrázek 4.1: Zjednodušené schéma rozpoznávače implementovaného v BSCORE.

Kapitola 5

Implementace dynamického dekodéru

V kapitole 2.7.4 je popsán princip dynamického dekodéru. Základní myšlenkou této práce není vytvořit plně dynamický dekodér (tj. dekodér dynamicky generující rozpoznávací síť pouze z akustických jednotek, výslovnostního slovníku a jazykového modelu), ale sestavit unigramovou rozpoznávací síť, a při dekódování s touto sítí modifikovat n-gramové pravděpodobnosti (pro n-gramy vyššího než prvního řádu) z načteného jazykového modelu. Jedná se tedy o dynamický dekodér, jehož dynamičnost spočívá v aplikování jazykového modelu až v průběhu rozpoznávání.

Pro dosažení výše popsaného dynamického dekodéru je nutné modifikovat algoritmus Token Passing použitý u statického dekodéru (viz podkapitola 5.1), sestavit unigramovou rozpoznávací síť, naimplementovat úložiště jazykového modelu (viz podkapitola 5.2) a implementovat modifikování n-gramových pravděpodobností v rámci načteného jazykového modelu a unigramové rozpoznávací sítě (viz podkapitola 5.3). Předchozí tři kroky stačí k získání dekodéru, jenž lze považovat za dynamický. Avšak pro vytvoření efektivního dynamického dekodéru, který přináší výrazné snížení paměťové náročnosti, je nutné implementovat vhodné optimalizace, popsané v podkapitole 5.4.

5.1 Modifikace algoritmu Token Passing

V kapitole 2.7.3 je popsán algoritmus Token Passing pro hledání minimální cesty v grafu na základě Viterbiho kritéria. Tento algoritmus se používá pro úsporu paměti při statickém dekódování. Pro dosažení dynamického dekódování je však nutné algoritmus Token Passing modifikovat.

Pokud se při algoritmu Token Passing potká více tokenů v jednom stavu, je ponechán jen ten s nejlepší cenou. My však u našeho dynamického dekodéru dekódujeme pouze s unigramovou rozpoznávací sítí (jenž reprezentuje prohledávaný graf), a v každém mezislovním přechodu chceme modifikovat pravděpodobnost (tzn. cenu) na základě n-gramových pravděpodobností nalezených ve vyhledávací struktuře uchovávající jazykový model. Pokud bychom tedy ponechávali v každém stavu pouze token s nejlepší cenou, mohla by v mezislovním přechodu modifikace této ceny vést k jejímu zhoršení natolik, že jiný token, jenž byl původně horší, by měl nakonec cenu lepší.

Řešením výše popsaného problému je stanovit kapacitu každého stavu na K tokenů. Pokud se tedy v jednom stavu setká N tokenů (přičemž $N > K$), je z nich vybráno K

s nejlepší cenou, které jsou uchovány a poslány dále. Nevýhodou je větší časová a paměťová náročnost zpracovávání tokenů ve stavech, jelikož je nutno uchovávat větší množství těchto tokenů, a jelikož hledání jednoho tokenu s nejlepší cenou je časově méně náročné než hledání K nejlepších z N (především protože tento výběr může být realizován při dekódování velice často).

Popis modifikovaného algoritmu Token Passing:

Symbolem $\Phi_{jk}(t)$ značme cenu k-tého nejlepšího tokenu ve stavu j v čase t a $\Psi_{jk}(t)$ identifikátor cesty tohoto tokenu. Poznamenejme, že předpokládáme ohodnocení hran rozpoznávací sítě záporným logaritmem pravděpodobnosti těchto hran, neboť algoritmus Token Passing slouží pro hledání nejkratší cesty v ohodnoceném orientovaném grafu.

1. **Inicializace v čase** $t = 1$

$$\Phi_{j1}(1) = \begin{cases} -\log b_j(o_1) - \log a_{1j} & \text{pro } j = 2, ..., N-1 \\ \infty & \text{pro } j = 1 \end{cases}$$

$$\Psi_{j1}(1) = 0 \qquad \text{pro } j = 1, .., N-1$$

2. **Pro všechny časy** $t = 2, ..., T$ **proveď**

   - Pro všechny stavy s_i, kde $i = 2, ..., N-1$ zkopíruj při změně časového kroku z $t-1$ na t všechny tokeny (tj. $k = 1, ..., K$) ze stavu s_i do všech stavů s_j, do kterých existuje přechod a zvětši cenu každého zkopírovaného tokenu:

     for $i = 2, ..., N-1$ **do**
       for $k = 1, ..., K$ **do**
         for $j = 2, ..., N-1$ **do**
           $\Phi_{ijk}(t) = (\Phi_{ik}(t-1) - \log a_{ij}) - \log b_j(o_t)$
           $\Psi_{ijk}(t) = \Psi_{ik}(t-1)$
         end for
       end for
     end for

     kde $\Phi_{ijk}(t)$ je nová cena k-tého tokenu zkopírovaného ze stavu s_i do s_j v čase t a $\Psi_{ijk}(t)$ je identifikátor této cesty.

   - Pro všechny stavy s_j, kde $j = 2, ..., N-1$ najdi ze všech tokenů zkopírovaných v čase t do stavu s_j K tokenů s nejlepším ohodnocením cesty $\Phi_{jk}(t)$ (pokud je celkový počet tokenů ve stavu s_j menší než K, ponech všechny) a všechny ostatní tokeny tohoto stavu odstraň. Pokud nalezený token při přechodu z času $t-1$ do času t vykonal mezislovní přechod, vytvoř pro něj nový záznam WLR:

     for $j = 2, ..., N$ **do**
       for $k = 1, ..., K$ **do**
         $m, p = \underset{i=2;k=1}{\overset{N-1;K}{\arg\min}} \ \Phi_{ijk}(t)$
         $\Phi_{jk}(t) = \Phi_{mjp}(t)$
         $\Psi_{jk}(t) = \Psi_{mjp}(t)$
         if $IsCrossWord(m, j)$ **then**
           $\Psi_{jk}(t) = CreateWLR(\Phi_{mp}(t-1), \Psi_{mp}(t-1), t, w(m))$

end if
delete $\Phi_{mjp}(t)$
delete $\Psi_{mjp}(t)$
end for
end for
for $i = 2, ..., N-1$ **do**
for $k = 1, ..., K$ **do**
delete $\Phi_{ijk}(t)$
delete $\Psi_{ijk}(t)$
end for
end for

kde $IsCrossWord(x, y)$ je splněno, pokud přechod ze stavu s_x do s_y je přechodem mezislovním, $w(x)$ je funkce vracející identifikátor slova s koncovým stavem s_x a funkce $CreateWLR(\Phi, \Psi, t, w)$ je funkce, která vrací identifikátor cesty rozšířený o nové slovo w v čase t.

- V rámci dynamického dekódování však funkce $CreateWLR$ také modifikuje cenu Φ na základě pravděpodobností z načteného jazykového modelu viz 5.3.

$$\Phi = \Phi - \textsc{GetProbability}(\Psi, 1) + \textsc{GetProbability}(\Psi, N) \tag{5.1}$$

kde

- $\textsc{GetProbability}(X,\ Y)$ je funkce popsaná v podkapitole 5.3, vracející podmíněnou n-gramovou pravděpodobnost pro náhradu unigramové pravděpodobnosti,
- parametry $(\Psi,\ 1)$ reprezentují poslední slovo tokenu uložené ve WLR a
- parametry $(\Psi,\ N)$ reprezentují N posledních slov tokenu uložených ve WLR.

- Odstraň staré tokeny z času $t-1$

for $i = 2, ..., N-1$ **do**
for $k = 1, ..., K$ **do**
delete $\Phi_{ik}(t-1)$
delete $\Psi_{ik}(t-1)$
end for
end for

3. **Výsledek**

- Token, jenž přijde do koncového neemitujícího stavu s_N s nejmenší hodnotou cesty udává index minimální cesty (tj. hledanou posloupnost slov) $\Psi_N(T^+)$ a její minimální cenu $\Phi_N(T^+)$:

$$m, p = \underset{i=2;k=1}{\overset{N-1;K}{\arg\min}} \ (\Phi_{ik}(T) - \log a_{iN})$$
$$\Phi_N(T^+) = \Phi_{mp}(T) - \log a_{mN}$$
$$\Psi_N(T^+) = CreateWLR(\Phi_{mp}(T), \Psi_{mp}(T), T, w(m))$$

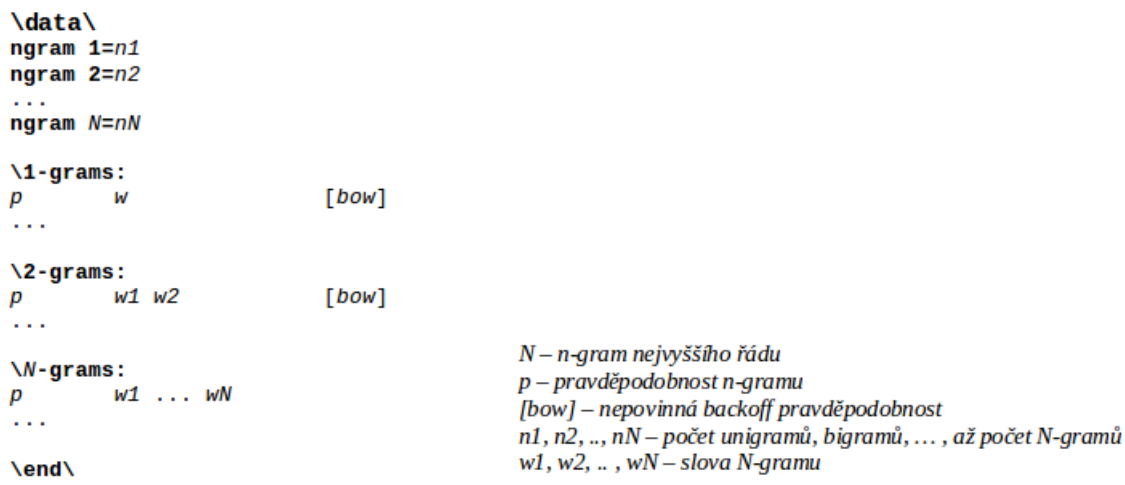
- Nejpravděpodobnější posloupnost slov pak získáme zpětným trasováním $\Psi_N(T^+)$.

Vzhledem ke změně použitého algoritmu je potenciálně možná i změna optimálního nastavení parametrů **beam pruning** a **live states pruning** popsaných v kapitole 2.7.3,

proto je nutné experimentální metodou nalézt optimum pro jejich nastavení. Podotkněme také, že především u beam pruningu je hledání optima zavádějící. Čím širší beam pruning je, tím více zůstane v systému tokenů, a tedy naroste časová i paměťová náročnost. Současně však širší beam pruning vede k vyšší přesnosti systému. Proto optimem zde myslíme kompromis mezi přesností a časovou a paměťovou náročností.

5.2 Modifikace rozpoznávací sítě a jazykového modelu

Původní dekodér firmy Phonexia popsaný v kapitole 4 využívá trigramovou transducerovou rozpoznávací síť, která je sestavena z 54 546 unigramů, 675 526 bigramů a 502 833 trigramů. Aby byly výsledky tohoto dekodéru porovnatelné s výsledky nově vzniklého dynamického dekodéru, je třeba používat tentýž jazykový model. K tomu využijeme **ARPA n-gram formát** viz [10], ve kterém máme tento jazykový model k dispozici, a ze kterého byla současná trigramová rozpoznávací síť sestavena. Na obrázku 5.1 převzatého z [10] lze vidět, jakým způsobem jsou zde n-gramy textově uloženy.



Obrázek 5.1: Textový ARPA formát pro uložení n-gramových pravděpodobností jazykového modelu.

Náš trigramový jazykový model v ARPA formátu jsme převedli na unigramový jazykový model odstraněním všech záznamů o bigramech a trigramech. Stejným procesem, kterým byla vytvořena z trigramového jazykového modelu trigramová rozpoznávací síť (využitím programů a skriptů firmy Phonexia), jsme vytvořili unigramovou rozpoznávací síť. Tato síť tvoří statickou část dekódování dynamického dekodéru a reprezentuje lexikální strom akustických jednotek.

K unigramové rozpoznávací síti potřebujeme také vhodné úložiště celého trigramového jazykového modelu. Jako jedinou počáteční podmínku jsme kladli schopnost tohoto úložiště načíst jazykový model z textového ARPA formátu a poskytování funkce vyhledávání n-gramu (tj. vrácení podmíněné pravděpodobností tohoto n-gramu). Jelikož jednotlivá slova v rámci jednoho n-gramu v textovém ARPA formátu jsou oddělena mezerami, můžeme pro vyhledávací účely celé toto sousloví uložit do hashovací tabulky jako textový řetězec. V rámci volby vhodné vyhledávací struktury byly vyzkoušeny `boost::unordered_map`,

`std::map` a `tr1/unordered_map`, přičemž finálně zvolenou byla `boost::unordered_map`. Rychlost a paměťové požadavky těchto struktur jsou shrnuty v tabulce 5.1.

Vyhledávací struktura	**Čas načtení** [s]	**100 000 vyhledání** [s]	**Paměť** [MB]
tr1/unordered_map	**18.5**	0.03	**1 530**
boost::unordered_map	19.5	**0.02**	1 585
std::map	29.1	0.07	1 840

Tabulka 5.1: Časová a paměťová náročnost vyhledávacích struktur pro textově uložený jazykový model.

5.3 Modifikace n-gramových podmíněných pravděpodobností

Pokud máme modifikovaný algoritmus Token Passing tak, aby bylo v každém stavu možné uchovat a poslat dále více tokenů (viz podkapitola 5.1), předchystanou unigramovou rozpoznávací síť a naimplementované úložiště pro vyhledávání n-gramových podmíněných pravděpodobností, je možné implementovat dynamické dekódování.

Dynamického dekódování dosáhneme modifikací pravděpodobností každého tokenu, jenž prochází slovním uzlem v unigramové rozpoznávací síti. Víme, že unigramová rozpoznávací síť k tokenu přičte logaritmus unigramové pravděpodobnosti aktuálně dekódovaného slova (jelikož počítáme s logaritmy pravděpodobností je zde operátor sčítání nikoli násobení, viz rovnice 2.14), a tedy pokud chceme „simulovat“ náš trigramový jazykový model, musíme logaritmus této unigramové pravděpodobnosti odečíst z celkového logaritmu pravděpodobnosti tokenu a místo ní přičíst logaritmus n-gramové pravděpodobnosti n-gramu nejvyššího nalezeného řádu z jazykového modelu.

Princip vyhledání vhodné n-gramové pravděpodobnosti, která má nahradit pravděpodobnost unigramovou, popisuje následující rekurzivní algoritmus, využívající úložiště S a reprezentaci n-gramu dvojicí (w, N), kde w jsou slova n-gramu a N je řád n-gramu (tj. počet těchto slov). Slova w jsou v reverzním pořadí (tj. vlevo je nejnovější dekódované slovo) pro snadnější využití slov uložených ve WLR.

Function GetProbability (w, N):

if $N \leq 0$ **then**
  return 0.0
end if
if $\mathrm{P}((w, N)) \in S$ **then**
  return $\mathrm{P}((w, N))$
end if
$t \leftarrow \mathrm{sufix}(w)$
$h \leftarrow \mathrm{prefix}(w)$
if $\mathrm{B}(t, N-1) \in S$ **then**
  return $\mathrm{B}(t, N-1) + \mathrm{GetProbability}(h, N-1)$
end if
return $\mathrm{GetProbability}(h, N-1)$

Kde

`sufix`(w) je funkce, která zkrátí slova w o první slovo,

`prefix`(w) je funkce, která zkrátí slova w o poslední slovo,

`P` je funkce udávající podmíněnou pravděpodobnost n-gramu jazykovým modelem,

`B` je funkce udávající back-off pravděpodobnost n-gramu jazykovým modelem.

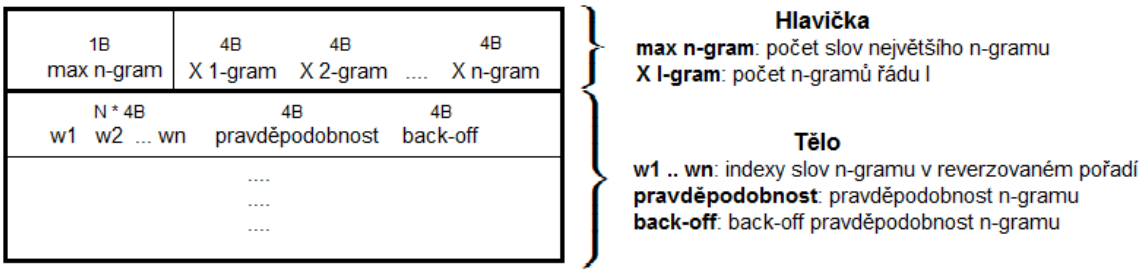
5.4 Optimalizace úložiště jazykového modelu

V této kapitole popíšeme implementace optimalizovaných řešení pro úložiště jazykového modelu. Implementace z podkapitol 5.4.1 a 5.4.2 ukládají úložiště jako binární nikoliv textový soubor. Slova v textovém zápise jsou zde nahrazena indexy slovníku z unigramové rozpoznávací sítě. Každý n-gram je v úložišti uložen reverzně, jelikož při dekódování máme k dispozici word link record (WLR) nejnovějšího slova, který ukazuje na WLR starších slov, a tedy toto pořadí nám umožňuje vyhnout se nutnosti reverzace každého vyhledávaného n-gramu při dekódování.

V podkapitole 5.4.3 je popsána optimalizace využitím softwarové cache paměti pro hledání n-gramových pravděpodobností určených k modifikaci za unigramovou pravděpodobnost. Tato optimalizace je použitelná pro obě navržené optimalizace popsané v 5.4.1 a 5.4.2.

5.4.1 Optimalizace hashovací strukturou

Optimalizace oproti primitivní textově orientované struktuře popsané v kapitole 5.2 spočívá v úspoře paměti díky nahrazení slov za indexy do slovníku unigramové rozpoznávací sítě, urychlení dekódování, neboť není nutné provádět reverzaci každého vyhledávaného n-gramu a urychlení načítání díky binárnímu uložení. Binární formát uložení vyhledávací struktury lze vidět na obrázku 5.2.



Obrázek 5.2: Binární formát optimalizovaného n-gramového úložiště používající `boost::unordered_map`.

Jako hashovací struktura byla taktéž využita `boost::unordered_map`. Jako klíč bylo však použito místo textového řetězce pole integerů (indexy slov ve slovníku). Hashování polem integerů bylo nutné doimplementovat, neboť není součástí `boost::unordered_map` (dodefinováním operátoru pro porovnání dvou polí integerů a postupný výpočet hashe tohoto pole).

Značnou nevýhodou této implementace je nutnost hashovat pomocí statického pole integerů fixní délky. Kdybychom totiž pro každý n-gram dynamicky alokovali ideální množství paměti, byl by výpočet velmi zpomalován neustálým voláním funkcí pro správu dynamicky alokované paměti (`malloc`, atd.). To však znamená, že pro uložení každého n-gramu je nutné

vytvořit statické pole o velikosti M integerů, kde M je řád nejvyššího n-gramu (tedy i unigram je uložen pomocí M integerů, kde $M-1$ položek je pouze zarovnání nulami).

Nutné množství paměti pro uložení této vyhledávací struktury lze vyjádřit následujícím vzorcem:

$$N(4M+8)+F \text{ Byte} \tag{5.2}$$

kde

N je počet n-gramů,

M je počet slov nejdelšího n-gramu (nejvyšší řád n-gramu),

F je paměť v bytech přidaná strukturou `boost::unordered_map`.

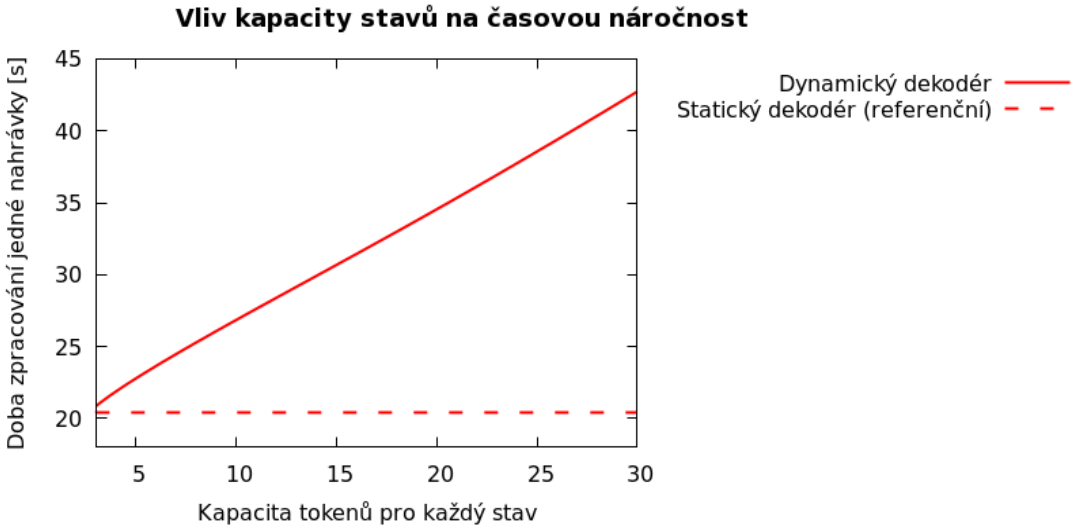
Na grafu z obrázku 5.3 lze vidět rozdíl paměťové náročnosti takto optimalizovaného dynamického dekodéru oproti referenčnímu statickému dekodéru a závislost paměťové náročnosti dynamického dekodéru na počtu uchovávaných tokenů v každém stavu (konstanta K z podkapitoly 5.1). Beam Pruning a maximální počet živých stavů je u obou zkoumaných dekodérů stejný (tedy $BP = 190$ a $MPS = 8\,000$).

Obrázek 5.3: Graf závislosti paměťové náročnosti na kapacitě stavů navržené optimalizace. Přerušovanou čarou je vyznačena pouze referenční hodnota statického dekodéru (viz 4.1), tj. kapacita stavů pro něj není měněna. Naměřené výsledky byly získány na základě podmínek popsaných v kapitole 3.3.

Další nevýhodou je stále pomalá inicializace vyhledávací struktury. Načítání z binárního souboru je sice rychlejší než z textového, jak tomu bylo u předchozí neoptimalizované varianty, avšak každý inicializovaný n-gram musí být při inicializaci vložen do `boost::unordered_map` funkcí `insert`, což vede k výraznému zpomalení oproti prostému načtení souboru do RAM paměti (viz další optimalizace v podkapitole 5.4.2).

Pro vyhledání n-gramové pravděpodobnosti, jež se použije pro modifikaci unigramové pravděpodobnosti z rozpoznávací sítě, je nutné použít algoritmu z 5.3, tedy nelze využít postupného prohledávání (viz další optimalizace v podkapitole 5.4.2).

Z grafu na obrázku 5.4 lze vidět, že časová náročnost takto optimalizovaného dynamického dekodéru je výrazně vyšší než u původního statického dekodéru, přičemž pro měření byly opět použity stejné hodnoty Beam Pruningu a maximálního počtu živých stavů.



Obrázek 5.4: Graf závislosti časové náročnosti na kapacitě stavů navržené optimalizace. Přerušovanou čarou je vyznačena pouze referenční hodnota statického dekodéru (viz 4.1), tj. kapacita stavů pro něj není měněna. Naměřené výsledky byly získány na základě podmínek popsaných v kapitole 3.3.

Pokud bychom chtěli dále optimalizovat tento přístup, nabízí se několik možností. Například implementace vlastní hashovací struktury by nám umožnila urychlit proces načítání z binárního souboru, jelikož bychom nemuseli strukturu inicializovat prostřednictvím funkce `boost::unordered_map::insert`, nýbrž pouhým načtením paměťového bloku. Vhodnou heuristikou by také bylo možné hashovat pomocí dynamicky alokovaného pole integerů, což by vedlo k úspoře paměti. Úspory paměti by bylo též možné dosáhnout vhodnou konverzí integerové reprezentace slov na jiný celočíselný typ o menším počtu bitů, neboť používáme 4 B integer což nám umožňuje kódovat až $2^{32} \approx 4.3 * 10^9$ slov, přičemž náš anglický jazykový model obsahuje jen okolo 60 000 slov, která by bylo možné kódovat pouze na 2 Bytech. Podobně i 4 B přesnost pro pravděpodobnosti by bylo možné redukovat například na 2 B s minimální očekávanou ztrátou přesnosti dekódování. Další možností optimalizace by bylo například postupné hashování po jednom slově, které by urychlilo výpočet z hlediska vyhledávání pravděpodobnosti, jež se použije jako náhrada unigramové pravděpodobnosti rozpoznávací sítě (viz 5.3).

Většinu navržených optimalizací zahrnuje optimalizace z následující kapitoly 5.4.2.

5.4.2 Optimalizace stromovou strukturou

Z důvodů nedostatků optimalizované struktury popsané v předchozí kapitole 5.4.1 byla navržena vyhledávací struktura, jež je předmětem této kapitoly. Na rozdíl od struktury z kapitoly 5.4.1 disponuje nová struktura následujícími vlastnostmi:

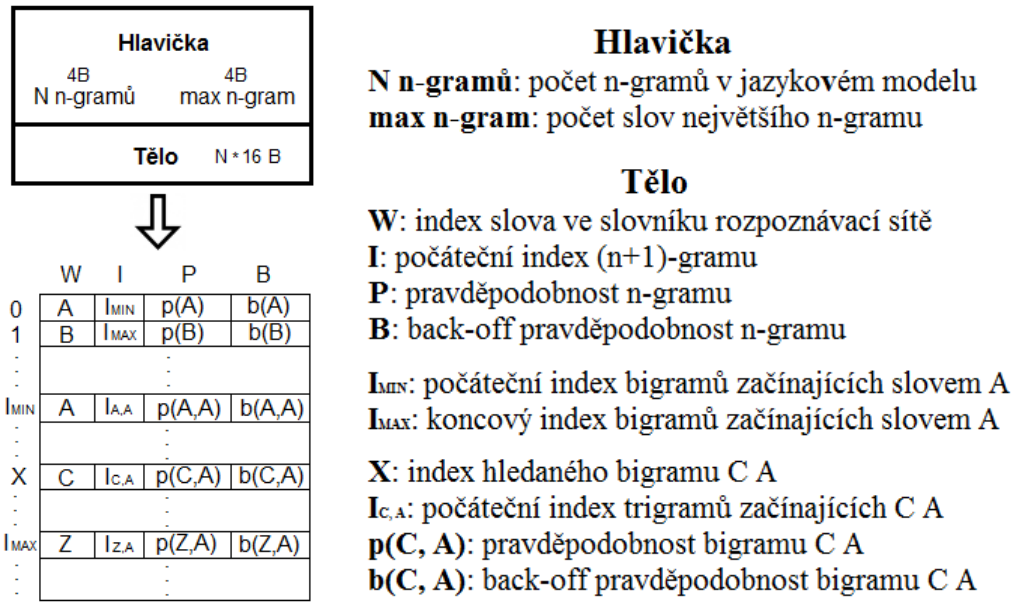
- načítání struktury z binárního souboru jako celého bloku paměti (není nutné každý načtený n-gram vkládat do vyhledávací struktury zvlášť),
- není založena na žádné externí vyhledávací struktuře, čímž redukuje přidanou paměť pro správu a správné fungování této struktury,
- umožňuje postupné vyhledávání n-gramové pravděpodobnosti, kterou se modifikuje unigramová pravděpodobnost z rozpoznávací sítě,
- unigramovou pravděpodobnost lze získat přímým přístupem do paměti,
- jednotlivé n-gramy jsou uloženy bez redundance paměti.

Na cílovou vyhledávací strukturu lze pohlížet jako na datový soubor, jehož každý řádek obsahuje informaci o slově, následujícím indexu, pravděpodobnosti n-gramu a back-off pravděpodobnosti. Pokud uvažujeme jazykový model obsahující N unigramů, odpovídá prvních N položek naší výhledávací struktury těmto unigramům. Slovník z rozpoznávací sítě nevyužívá všechny indexy slov (tj. mezi použitými indexy existují mezery), používáme proto pomocné pole, které mapuje tyto indexy slov na nové indexy, mezi kterými tyto mezery neexistují. Využitím těchto nových indexů pak můžeme přistupovat na položky unigramů přímým přístupem do paměti úložiště.

Pokud chceme nalézt n-gram vyššího než prvního řádu, dopátráme se jej v navržené vyhledávací struktuře hierarchickým prohledáváním. Uvažujme například hledání bigramu `A C` (odpovídá bigramu `C A` v ARPA formátu, vzhledem k reverznímu pořadí slov, jak je popsáno výše). Nejprve přímým přístupem do paměti přistoupíme na index A, kde nalezneme počáteční index I_{MIN} a na indexu $A+1$ koncový index I_{MAX} všech bigramů, jenž začínají slovem `A`. Všechny záznamy mezi indexy I_{MIN} a I_{MAX} jsou vždy seřazeny, a tedy vyhledat záznam bigramu `A C` můžeme binárním vyhledáváním (nalezením záznamu se slovem `C`). Pokud bychom chtěli vyhledat n-gram vyššího řádu, budeme postupovat stejným způsobem dále (tj. počáteční index trigramů nalezneme na nalezené položce AC a koncový na $(AC)+1$, binárním prohledáváním pak můžeme najít vyhledávaný n-gram). Příklad vyhledávání bigramu `A C` a uložení n-gramů v paměti lze vidět na obrázku 5.5.

Pokud se pro daný n-gram A v jazykovém modelu nevyskytuje žádný n-gram vyššího řádu, jemuž je A prefixem (tj. neexistuje žádný „$(n+1)$-gram“), je zde hodnota indexu uložena jako záporné číslo. Absolutní hodnota tohoto indexu potom reprezentuje maximální index pro n-gramy na indexu $A-1$.

Problém by též nastal, pokud by součástí jazykového modelu byl n-gram řádu n, pro který neexistuje n-gram řádu $n-1$, který je mu prefixem ve smyslu naší vyhledávací struktury. Jelikož takové n-gramy existují, je nutné jazykový model rozšířit o n-gramy nižšího řádu tak, aby ve výsledném jazykovém modelu neexistoval n-gram, jehož libovolně dlouhý prefix není součástí jazykového modelu.



Obrázek 5.5: Formát uložení a způsob vyhledávání n-gramových pravděpodobností optimalizovanou stromovou strukturou.

Algoritmus vytvoření popsané vyhledávací struktury

Uvažujme na vstupu jazykový model LM v textovém ARPA formátu, z něhož chceme vytvořit binární reprezentaci vyhledávací struktury SS, jež je popsána v této kapitole, a uložit ji do výstupního souboru F. Současně máme k dispozici statickou unigramovou rozpoznávací síť (vytvořenou z LM), jež slouží pro převod textově reprezentovaných slov na celočíselný index do slovníku rozpoznávací sítě.

Výstupem algoritmu je binárně uložený soubor obsahující vyhledávací strukturu tak, jak je popsáno na obrázku 5.5.

Proces převodu z LM na naši vyhledávací strukturu popisuje následující algoritmus.

1. Převedeme každé slovo w každého n-gramu g našeho jazykového modelu LM na jeho integerovou reprezentaci a provedeme reverzaci pořadí těchto n-gramů.

   for $\forall g \in LM$ **do**
   $g \leftarrow \text{rev}(g)$
   for $\forall w \in g$ **do**
   $w \leftarrow \text{w2i}(w)$
   end for
   end for

   Kde funkce `rev`*(array)* vrací vstupní pole *array* v opačném pořadí. Funkce `w2i` konvertuje textovou podobu slova na integerovou reprezentaci (využívá pomocného mapovacího pole, jenž je součástí implementace rozpoznávací sítě).

2. Rozšíříme jazykový model o n-gramy, které jsou prefixy jiných n-gramů, a to následujícím způsobem. Vytvoříme pomocnou hashovací strukturu HT, jež bude uchovávat informace o všech n-gramech, které jsou součástí jazykového modelu. Poté pro každý

n-gram řádu n zjistíme, zda jeho prefix délky $n-1$ je n-gramem v LM (předpokládejme, že LM je seřazeno podle řádu n-gramů), pokud ne, je tento n-gram prefixu vytvořen, a je mu správně dopočítána pravděpodobnost na základě metody popsané v 5.3.

for $\forall g \in LM$ **do**
  $HT\,[g] \leftarrow [\mathrm{P}(g), \mathrm{B}(g)]$
  if prefix$(g) \notin HT$ **and** $|g| > 2$ **then**
    AddNGrams$(HT, \text{prefix}(g))$
  end if
end for

Function AddNGrams (HT,g):

$\mathrm{P}(g) \leftarrow$ GetProbability$(g, |g|)$ (viz algoritmus z kapitoly 5.3)
$\mathrm{B}(g) \leftarrow 0.0$
$HT\,[g] \leftarrow [\mathrm{P}(g), \mathrm{B}(g)]$
$LM\,[g] \leftarrow [\mathrm{P}(g), \mathrm{B}(g)]$
if prefix$(g) \notin HT$ **then**
  AddNGrams$(HT, \text{prefix}(g))$
end if

Kde `prefix`(g) je funkce, která vrací n-gram o jedna nižšího řádu než je g (tj. bez posledního slova). Funkce `GetProbability` vrací pravděpodobnost na základě algoritmu popsaném v kapitole 5.3.

3. Seřadíme všechny n-gramy z LM podle jejich číselné reprezentace slov, přičemž n-gramy nižšího řádu jsou při porovnávání považovány za menší než n-gramy vyššího řádu (viz obrázek 5.6).

0
1
2
0 1
1 2
0 1 2
0 2 1
⋮

Obrázek 5.6: Příklad seřazení n-gramů podle číselné reprezentace slov.

4. Konstrukce naší vyhledávací struktury SS z takto seřazeného jazykového modelu LM, rozšířeného o „prefixové“ n-gramy, čítající celkem $|LM|$ n-gramů, je provedena následovně:

for $i = 0, ..., |LM| - 1$ **do**
  $SS_i.W \leftarrow$ LastWord(LM_i)
  $SS_i.P \leftarrow \mathrm{P}(LM_i)$
  $SS_i.B \leftarrow \mathrm{B}(LM_i)$
  $EQ \leftarrow \{j : i < j < |LM|, LM_i = \text{prefix}(LM_j)\}$

$$I_{i,min} \leftarrow \begin{cases} \min(EQ) & \text{pro } EQ \neq \emptyset \\ -1 & \text{pro } EQ = \emptyset \end{cases}$$
$$I_{i,max} \leftarrow \begin{cases} \max(EQ) & \text{pro } EQ \neq \emptyset \\ 1 & \text{pro } EQ = \emptyset \end{cases}$$
if $I_{i,min} = -1$ **then**
$SS_i.I \leftarrow -I_{i-1,max}$
else
$SS_i.I \leftarrow I_{i,min}$
end if
end for

kde

- `LastWord` je funkce vracející poslední slovo n-gramu,
- SS_i resp. LM_i reprezentují i-tou položku SS resp. LM,
- $SS_i.W$, $SS_i.I$, $SS_i.P$, $SS_i.B$ odpovídají sloupcům W, I, P, B z obrázku 5.5,
- `P` a `B` jsou funkce vracející pravděpodobnosti n-gramu,
- $I_{i,min}$ a $I_{i,max}$ je minimální resp. maximální index n-gramů řádu n s prefixem LM_i a délkou $n-1$.

5. Binárně uložíme vytvořenou vyhledávací strukturu SS do výstupního souboru F včetně hlavičky obsahující celkový počet n-gramů $|SS|$ a řád největšího n-gramu $\max(\{|g|, g \in LM\})$.
   BinarySave(F, $|SS|$)
   BinarySave(F, $\max(\{|g|, g \in LM\})$)
   BinarySave(F, SS)

Kde funkce `BinarySave`(F, D) binárně uloží data D do souboru F.

Algoritmus vyhledání cílové n-gramové pravděpodobnosti

V kapitole 5.3 je popsán algoritmus výběru n-gramové pravděpodobnosti, kterou bude nahrazena unigramová pravděpodobnost, získaná dekódováním ve statické unigramové rozpoznávací síti. Vzhledem k rozdílnému uložení pravděpodobností n-gramů jazykového modelu ve struktuře popsané v této kapitole, musí být změněn i algoritmus výběru této pravděpodobnosti.

Modifikovaný algoritmus využívá pro nalezení nejvhodnější pravděpodobnosti pro n-gram $g = (w, N)$, kde w jsou slova n-gramu a N je řád tohoto n-gramu, postupného prohledávání vyhledávací struktury SS a uchovávání již nalezených pravděpodobností. Proto není nutné implementovat algoritmus rekurzivně, jako tomu bylo u algoritmu z kapitoly 5.3.

$P \leftarrow \{\mathrm{P}(g) | g \in \mathrm{GetProbs}(SS, w, N)\}$
$B \leftarrow \{\mathrm{B}(g) | g \in \mathrm{GetProbs}(SS, \mathrm{sufix}(w), N-1)\}$
if $|P| > |B|$ **then**
return $P_{|P|}$
end if

while $|B| > |P|$ **do**
  $B_{|B|-1} \leftarrow B_{|B|-1} + B_{|B|}$
  $B \leftarrow \text{prefix}(B)$
end while

return $P_{|P|} + B_{|B|}$

Kde funkce GetProbs(SS, w, N) vrací všechny postupně nalezené pravděpodobnosti z LM pro N slov ve w. Funkce `sufix` vrací pole slov bez prvního slova. $|S|$ je obecně značen počet prvků struktury S a S_i značí i-tou položku struktury S (tedy $S_{|S|}$ značí poslední položku struktury S).

Function GetProbs (SS,w,N):

$i \leftarrow 1$
$g_i \leftarrow \{\text{P: } SS_{w_i}.P,\ \text{B: } SS_{w_i}.B\}$
$min \leftarrow SS_{w_i}.I$
$max \leftarrow |SS_{w_i+1}.I|$

while $min \geq 0 \wedge i < N$ **do**
  $i \leftarrow i + 1$
  $k \leftarrow \text{BinarySearch}(SS, w_i, min, max)$
  if $k \geq 0$ **then**
    $g_i \leftarrow \{\text{P: } SS_k.P,\ \text{B: } SS_k.B\}$
    $min \leftarrow SS_k.I$
    $max \leftarrow |SS_{k+1}.I|$
  else
    break
  end if
end while
return g

Funkce BinarySearch(SS, w, min, max) provádí binární hledání prvku se slovem w ve struktuře SS mezi indexy min a max. Vrací index této položky nebo hodnotu -1, pokud prvek nebyl nalezen.

Function BinarySearch (SS,w,min,max):

while $min \leq max$ **do**
  $m \leftarrow \frac{max-min}{2}$
  if $SS_m.W < w$ **then**
    $min \leftarrow m + 1$
  else if $SS_m.W > w$ **then**
    $max \leftarrow m - 1$
  else
    return m
  end if
end while
return -1

Pro každý n-gram máme v naší struktuře jeden záznam obsahující poslední slovo, index následujících n-gramů vyššího řádu a dvě hodnoty typu `float` pro pravděpodobnosti.

Výsledná požadovaná paměť je tedy dána vztahem

$$N * 16\,\mathrm{B} \tag{5.3}$$

kde N je počet n-gramových záznamů jazykového modelu. Pokud porovnáme paměťové požadavky naší optimalizované struktury se strukturou z kapitoly 5.4.1 (rovnice 5.2), můžeme určit, kolika násobek celkově využité paměti ušetříme, pomocí následujícího vztahu:

$$\frac{N(4M+8)+F}{N*16} = \frac{4*(M+2)}{16} + \frac{F}{16*N} = \frac{M+2}{4} + \frac{F}{16*N} \tag{5.4}$$

Poznamenejme, že pracovat s menším než bigramovým jazykovým modelem postrádá smysl kvůli nízké úspěšnosti dekódování (viz tabulka 5.2) a už pro $M = 2$ je paměťová úspora zaručena o $\frac{F}{16N}$. Nejčastěji přitom pracujeme s trigramovými jazykovými modely ($M = 3$), což nám i pokud opomeneme $\frac{F}{16N}$ garantuje úsporu alespoň 20 % celkové paměti. Reálná úspora paměti je pak obvykle výrazně větší, neboť $\frac{F}{16N}$ má nemalý význam. V kapitole 6 je pak také zjištěno, že využití dynamického dekodéru s 5-gramovým jazykovým modelem má potenciál lepších výsledků, a tedy paměťová úspora i se zanedbáním $\frac{F}{16N}$ je potom téměř 60 % celkové paměti.

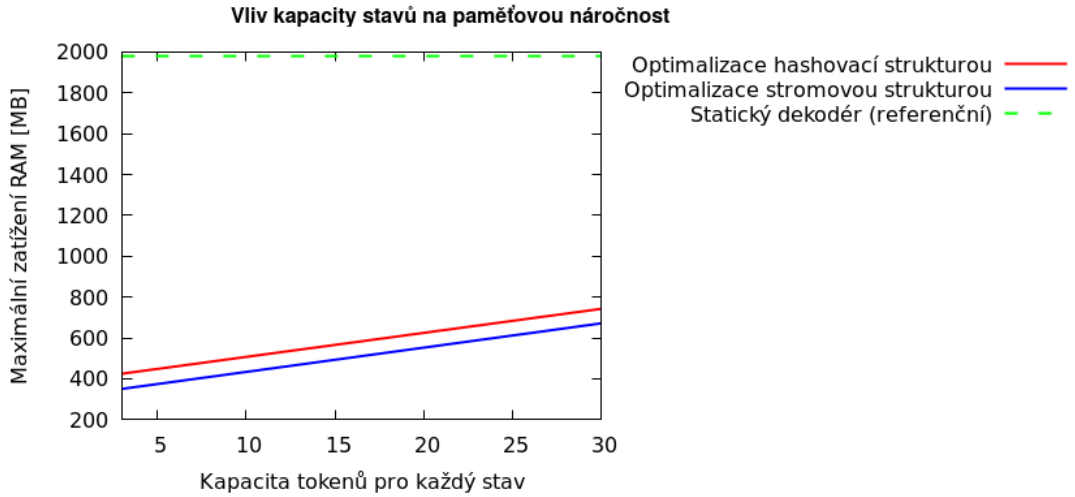
CC1 (WA)	**CC2** (WA)	**RZ** (WA)	**N01** (WA)
62.4 %	44.7 %	51.6 %	65.6 %

Tabulka 5.2: Úspěšnost dekódování statickým dekodérem s unigramovou rozpoznávací sítí za podmínek z kapitoly 3.3. Průměrný rozdíl v úspěšnosti oproti referenčnímu trigramovému jazykovému modelu zde činí 10.83 %.

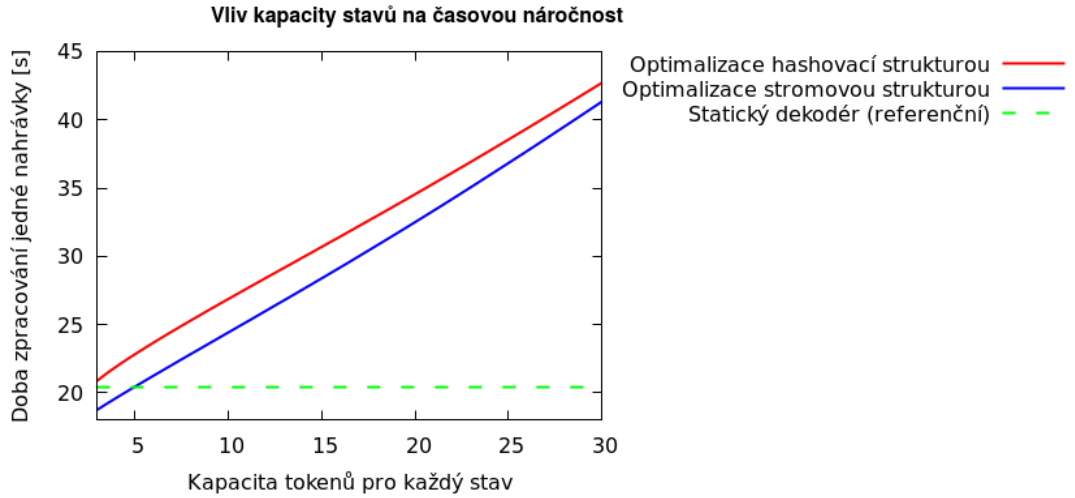
Paměťovou úsporu oproti optimalizaci hashovací strukturou z kapitoly 5.4.1 lze vidět na grafu z obrázku 5.7, na kterém je také zachycena závislost na kapacitě jednotlivých stavů rozpoznávací sítě. Všechny zkoumané konfigurace byly ohodnoceny pro stejnou hodnotu Beam Pruningu i maximálního počtu živých stavů.

Optimalizace navržená v této kapitole přináší nejen zlepšení paměťové náročnosti viz 5.7, ale vzhledem k možnosti postupného vyhledávání cílové n-gramové pravděpodobnosti, kterou bude nahrazena unigramová pravděpodobnost z rozpoznávací sítě, také zlepšení náročnosti časové. Zlepšení je zachyceno v grafu na obrázku 5.8.

Dalšími možnostmi optimalizace této vyhledávací struktury je převod integerové reprezentace slov na jiný celočíselný typ o menším počtu bitů, jak již bylo uvedeno v kapitole 5.4.1. Nevýhodou změny tohoto typu můžou být budoucí problémy při aplikaci úložiště na jazyk s větším počtem slov jako je například čeština. V kapitole 5.4.1 je taktéž již uvedena možnost zmenšit paměťový prostor určený pro uložení n-gramových pravděpodobností s minimální očekávanou ztrátou přesnosti dekódování.



Obrázek 5.7: Graf závislosti paměťové náročnosti na kapacitě stavů navržené optimalizace stromovou strukturou ve srovnání s optimalizací hashovací strukturou z kapitoly 5.4.1. Přerušovanou čarou je vyznačena pouze referenční hodnota statického dekodéru (viz 4.1), tj. kapacita stavů pro něj není měněna. Naměřené výsledky byly získány na základě podmínek popsaných v kapitole 3.3.



Obrázek 5.8: Graf závislosti časové náročnosti na kapacitě stavů navržené optimalizace stromovou strukturou ve srovnání s optimalizací hashovací strukturou z kapitoly 5.4.1. Přerušovanou čarou je vyznačena pouze referenční hodnota statického dekodéru (viz 4.1), tj. kapacita stavů pro něj není měněna. Naměřené výsledky byly získány na základě podmínek popsaných v kapitole 3.3.

5.4.3 Optimalizace cache pamětí

Dynamické dekódování umožňuje značnou úsporu paměti ve srovnání se statickým dekódováním využívajícím velkou rozpoznávací síť. Za cenu této úspory je ovšem dekódování zpomaleno vlivem správy většího počtu tokenů v modifikovaném algoritmu Token Passing, popsaném v kapitole 5.1, a nutností modifikovat unigramové pravděpodobnosti na základě pravděpodobností hledaných ve vyhledávací struktuře principem popsaným v 5.3.

Využitím optimalizované stromové struktury, popsané v kapitole 5.4.2, dojde sice k výraznému zlepšení vlastností implementace z hlediska paměťové náročnosti, z hlediska časové náročnosti je zde však stále prostor k optimalizacím.

Z analýzy chování dosavadního dynamického dekodéru používajícího optimalizovanou stromovou strukturu popsanou v kapitole 5.4.2 vyplývá, že proces výpočtu pravděpodobnosti (tj. vyhledávání v naší struktuře) pro modifikaci unigramové pravděpodobnosti zabírá 5-10 % doby rozpoznávání řeči. Proto optimalizovat právě tuto část dekódování má velký potenciál úspory z hlediska časové složitosti.

Za výše popsaným účelem optimalizace byl navržen koncept softwarového **cachování** již nalezených pravděpodobností (pomocí algoritmu popsaném v 5.4.2), jelikož experimentálně bylo zjištěno, že je-li vyhledávána pravděpodobnost n-gramu g, jež je součástí nějakého tokenu, je vysoce pravděpodobné, že pravděpodobnost pro tentýž n-gram bude brzy vyhledávána znovu.

Pro cachování je využito podobné hashovací struktury jako je použita v kapitole 5.4.1. Vyhledávané n-gramy jsou klíči této hashovací struktury. Pro každý klíč obsahuje hashovací tabulka iterátor do prioritní fronty (tj. ukazuje na záznam pro daný n-gram). Tato fronta pak určuje pořadí n-gramů v rámci cachování a obsahuje podmíněné pravděpodobnosti jednotlivých n-gramů.

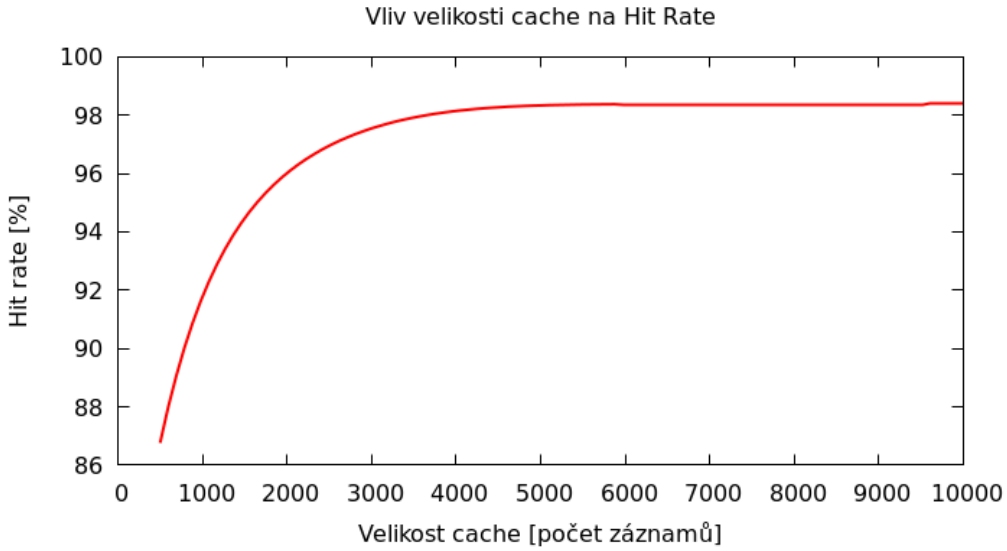
Algoritmus dynamického dekódování poté změníme tak, že pokud hledáme podmíněnou pravděpodobnost pro modifikaci unigramové pravděpodobnosti n-gramu g, nejprve se ji pokusíme nalézt v cache paměti, a teprve v případě neúspěchu ji určíme algoritmem popsaným v kapitole 5.4.2. Pokud n-gram není v cache paměti, tak jej přidáme, takže pokud v příštím kroku bude opět hledána podmíněná pravděpodobnost n-gramu g, již bude nalezena. Při přidávání je prvek současně přidán na začátek fronty cache paměti, která udržuje informaci o pořadí jednotlivých n-gramů v cache. Pokud kapacita cache při přidávání přeteče, je prvek z konce fronty odstraněn včetně jeho obrazu v hashovací tabulce. Pokud n-gram g při hledání v cache paměti nalezneme, přesuneme jej v rámci fronty pořadí na začátek a vrátíme nalezenou pravděpodobnost.

Implementace softwarové cache paměti je realizována pomocí `boost::unordered_map` v jazyce C++, kde jako klíč je použita třída `SHashKey` obsahující statické pole integerů (s maximálním omezením na 5-gramy). Nad touto třídou je také definován operátor porovnání a výpočtu hashe. Hodnotou každého záznamu v mapě je iterátor do listu `std::list`, jenž reprezentuje frontu uchovávající pořadí jednotlivých záznamů. Položkami tohoto listu jsou pak jednotlivé záznamy cache, tedy `std::pair<SHashKey, float>`.

Implementace cache paměti funguje následovně. Při přidávání (funkce `Insert`) předpokládáme, že prvek ve struktuře není, jelikož při naší realizaci je funkce `Insert` vždy volána až po neúspěšném hledání funkcí `Get` a následném nalezení hodnoty v jazykovém modelu metodou z kapitoly 5.4.2. Tím ušetříme zbytečné druhé hledání v hash tabulce. Přidávaný prvek tedy vložíme na začátek fronty a iterátor ukazující na tento prvek vložíme do hash tabulky. Pokud přidáním nového prvku do cache došlo k překročení kapacity fronty, je z hash tabulky odstraněn nejstarší záznam daný klíčem posledního prvku fronty.

Při vyhledávání prvku funkcí `Get` je hledaný n-gram g použit jako klíč pro hashovací tabulku. Pokud dojde k jeho nalezení, je získán iterátor na položku do fronty záznamů, je vrácena pravděpodobnost zde uložená, a funkcí `splice` (kontejneru `std::list`) je prvek přesunut na začátek fronty. V případě že prvek nalezen není je pouze vrácena hodnota `false`.

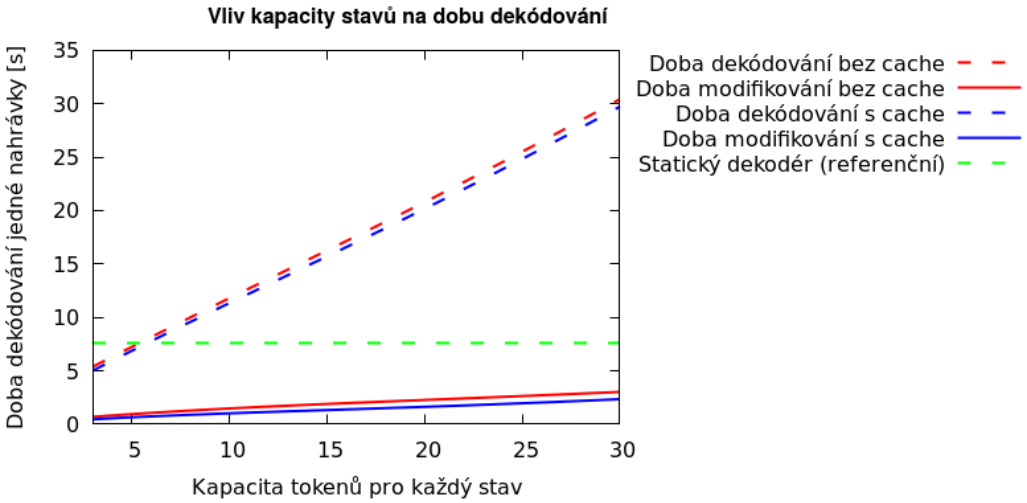
Pro vhodné nastavení výše popsaného cashování je nutné určit kapacitu cash paměti, tj. kolik záznamů o pravděpodobnostech bude možné uložit, než se cache začne od konce fronty promazávat. Ideální je nastavit takovou velikost cash paměti, pro kterou by již další zvětšení nepřineslo významné zvýšení úspěšnosti nalezení – tzv. **hit rate**. K tomu nám poslouží následující graf z obrázku 5.9.



Obrázek 5.9: Graf závislosti hit rate na velikosti cache paměti pro původní trigramový jazykový model.

Z grafu 5.9 lze vidět, že pro velikost cache větší než 6 000 je hit rate takřka konstantní. Je nutné si však také položit otázku, jak velké množství paměti bude takto velká cache zabírat. Jeden záznam v cache paměti odpovídá pěti integerům pro reprezentaci n-gramu, iterátoru do fronty a pravděpodobnosti typu float. K celkovému množství je také nutné počítat s pamětí kterou používá `std::list` a `boost::unordered_map`. Experimentálně bylo zjištěno, že i cache o velikosti 10 000 prvků zabírá v paměti něco málo přes 1 MB, což je paměť zanedbatelná v porovnání s paměťovou náročností celého dekodéru.

Na grafu z obrázku 5.10 lze nalézt časové srovnání dynamického dekodéru založeného na druhé navržené optimalizaci v kapitole 5.4.2 a používající softwarové cachování se stejným dekodérem, jenž cachování nepoužívá. Můžeme si všimnout, že s rostoucí velikostí kapacity stavů roste také časová úspora. Byť se tento rozdíl nemusí zdát v kontextu celého rozpoznávače významný, zrychluje vyhledávání pravděpodobností určených k modifikaci v případě systému s kapacitou 15 tokenů na stav o 31.5 % (tedy o 0.60 s na nahrávce o délce 30.76 s).



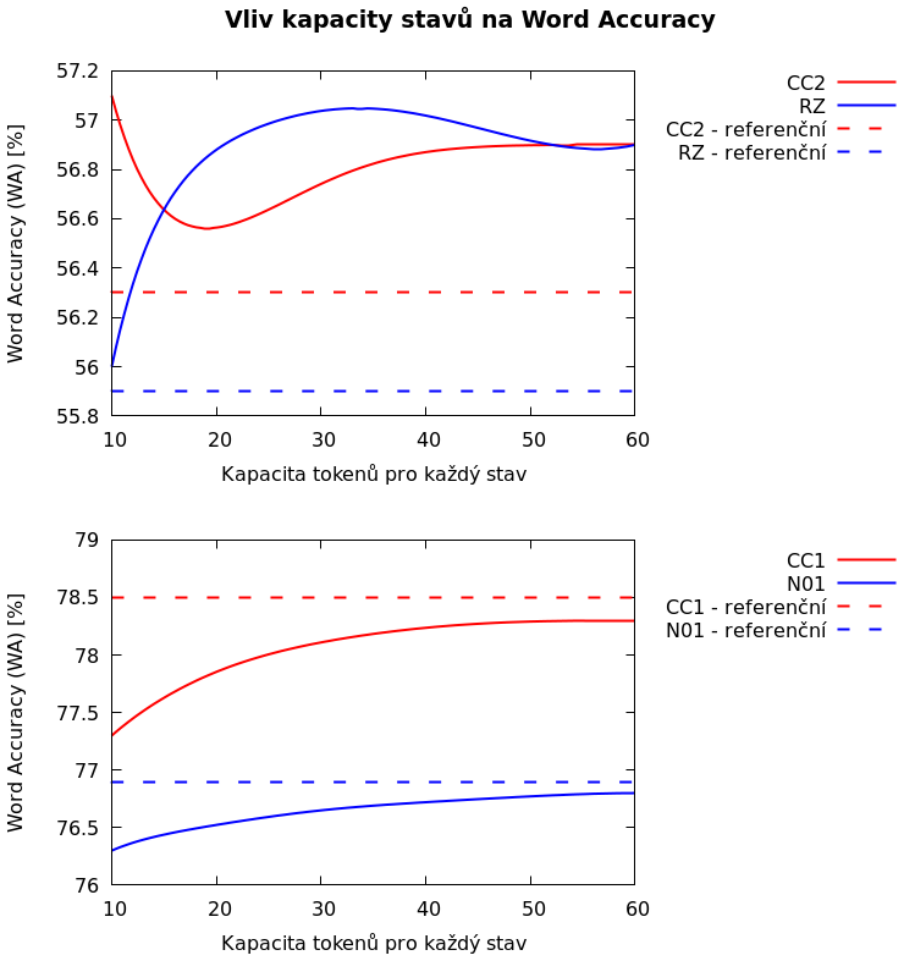
Obrázek 5.10: Graf závislosti rychlosti dekódování na kapacitě stavů pro dynamický dekodér využívající optimalizovanou stromovou strukturu z kapitoly 5.4.2 spolu se softwarovou cache pamětí z této kapitoly. Referenční hodnotou statického dekodéru zde myslíme hodnotu z tabulky 4.1, přičemž kapacita stavů nebyla pro statický dekodér měněna. Naměřené výsledky byly získány na základě podmínek popsaných v kapitole 3.3.

5.5 Experimenty a výsledky

V předchozích podkapitolách byly uvedeny nutné modifikace statického dekodéru pro získání dekodéru dynamického, včetně dvou navržených optimalizovaných struktur pro uložení jazykového modelu a návrhu softwarového cachování pro zrychlení vyhledávání podmíněných pravděpodobností n-gramů. Jelikož lze předpokládat rapidní změnu chování dekodéru při aplikaci všech navržených modifikací, bylo provedeno několik různých experimentů pro získání srovnání původního a nového dekodéru. Dále byly lazeny hodnoty parametrů jako je beam pruning, maximální počet živých stavů a kapacita jednotlivých stavů.

Hlavním zkoumaným atributem každého rozpoznávače je přesnost dekódování (*word accuracy*). Optimálním výsledkem z hlediska zkoumání přesnosti je pro dynamický dekodér dosažení stejných výsledků jako u dekodéru statického. Lze však předpokládat, že úspěšnost našeho dynamického dekodéru bude záviset na kapacitě jednotlivých stavů. V případě, že bychom kapacitu stavů nezvětšili, nemělo by modifikování unigramových pravděpodobností z externího jazykového modelu žádný význam. Graf závislosti kapacity stavů na přesnost rozpoznávače s dynamickým dekodérem lze vidět na obrázku 5.11.

Z důvodu přehlednosti byl graf výsledků rozdělen na dva grafy. Jak můžeme vidět, pro datasety **CC1** a **N01** je přesnost rozpoznávače výrazně lepší než pro datové sady **CC2** a **RZ**. Taktéž si všimněme, že pro datové sady CC1 a N01 úspěšnost vždy pro vyšší kapacitu stavů roste, a že úspěšnost nikdy nepřekoná původní statický dekodér, což je očekávané chování. Pro datasety CC2 a RZ je však závislost na kapacitě stavů poměrně chaotická, a dynamický dekodér pro tyto datové sady generuje lepší výsledky, což je v rozporu s očekávaným chováním. Dosažení lepších výsledků, než které vykazuje statický dekodér, je možné vysvětlit rozdílným aplikováním prořezávacích technik (pruningu) při statickém a dynamickém dekódování (tzn. přežijí tokeny, které u statického dekodéru byly prořezány).



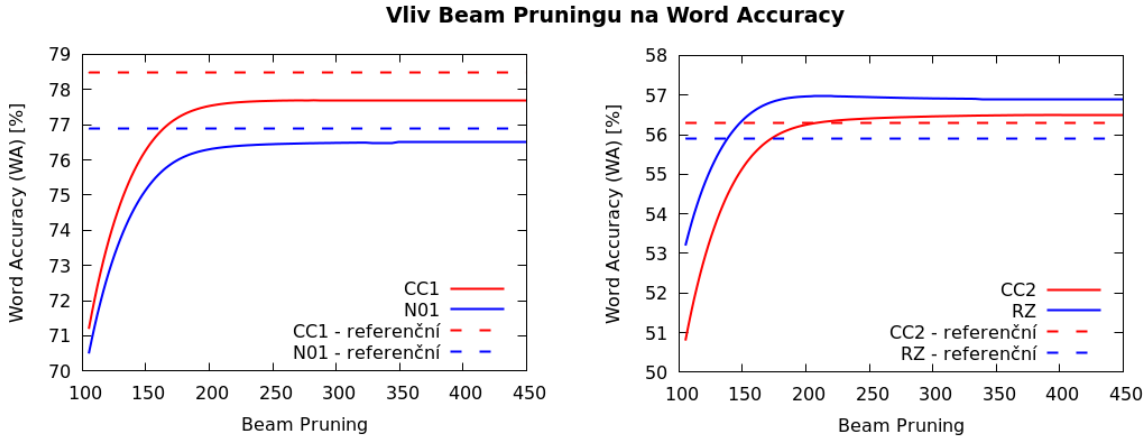
Obrázek 5.11: Graf závislosti word accuracy na kapacitě stavů. Přerušovanou čarou jsou zde uvedeny referenční hodnoty statického dekodéru (viz tabulka 4.1), pro které nebyla kapacita stavů měněna.

Nerostoucí části grafu pro datové sady CC2 a RZ mohou být způsobeny rekombinací tokenů. Tyto nesrovnalosti s očekávaným chováním však budou ještě předmětem dalšího zkoumání.

Jako vhodné nastavení kapacity stavů bychom tedy pravděpodobně zvolili hodnotu v rozmezí 40 až 60. Avšak pokud vezmeme v úvahu časovou náročnost takovéto konfigurace viz graf 5.10, pro kapacitu $k = 40$ by dekódování testovací 30.76 sekund dlouhé nahrávky trvalo více než 50 sekund, což značně nevyhovuje požadavkům pro schopnost rozpoznávače fungovat v reálném čase (tato vlastnost je samozřejmě subjektivní vzhledem ke stroji, na kterém je rozpoznávač spuštěn). Pro náš testovací stroj dosáhneme fungování v reálnem čase pro kapacitu $k = 15$, kdy naši testovací nahrávku systém zpracuje za 28.62 s (při konfiguraci dekodéru využívající optimalizovanou strukturu z kapitoly 5.4.2 a cachování z kapitoly 5.4.3).

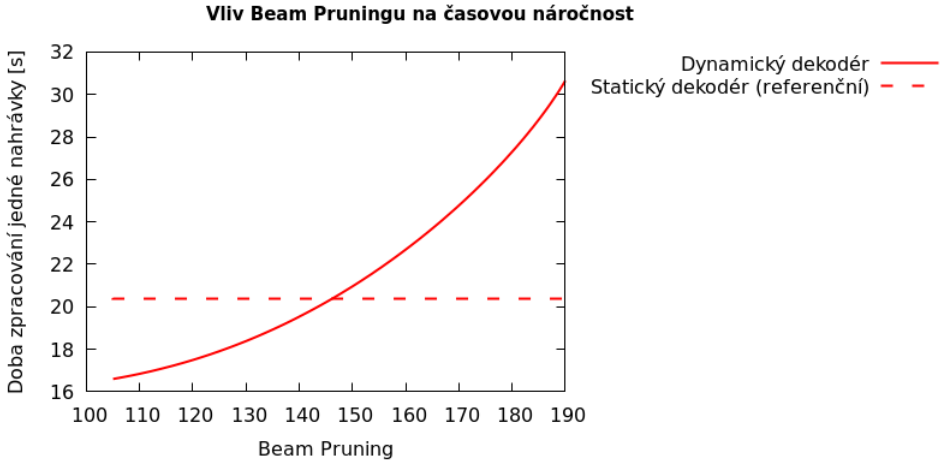
Parametry beam pruning a maximální počet živých stavů byly pro původní statický dekodér zvoleny experimentálně na $BP = 190$ a $MS = 8000$. Poznamenejme, že čím větší tyto

hodnoty jsou, tím přesnější lze výsledný rozpoznávač očekávat, avšak za cenu zvýšení časové a paměťové náročnosti z důvodu uchovávání většího počtu tokenů. Proto vhodné zvolení těchto hodnot lze chápat jako hledání kompromisu mezi přesností, rychlostí a potřebnou operační pamětí. Jelikož však není možné zcela předvídat, zda by změna těchto hodnot nevedla k výhodnějšímu nastavení, byly tyto experimenty provedeny znovu pro dynamický dekodér. Na grafu 5.12 lze vidět vliv Beam Pruningu na přesnost rozpoznávání.



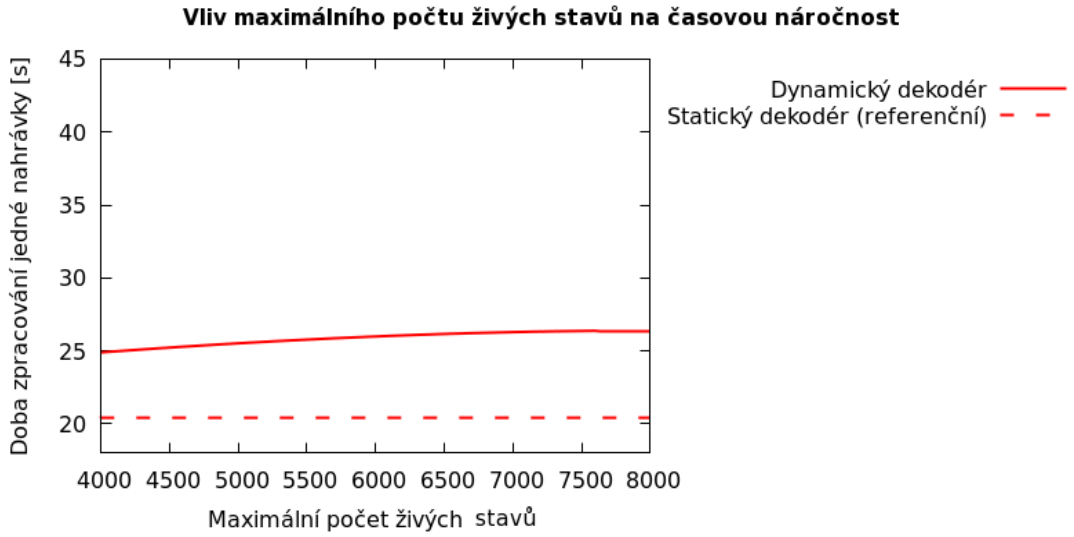
Obrázek 5.12: Graf závislosti word accuracy na beam pruningu. Přerušovanou čarou je vyznačena referenční hodnota získaná skórováním původního statického dekodéru. Hodnota beam pruningu zde není pro statický dekodér měněna.

Z grafu 5.12 lze usoudit, že hodnota beam pruningu $BP = 190$ se jeví jako velmi vhodně nastavená. Pokud však zvážíme i graf závislosti doby zpracování naší nahrávky na beam pruningu (obrázek 5.13) viz podmínky uvedené v kapitole 3.3, lze uvažovat o jejím nepatrném snížení, například na $BP = 175$, kde pokles přesnosti je poměrně zanedbatelný vzhledem k uspořenému času.



Obrázek 5.13: Graf závislosti doby rozpoznávání na beam pruningu. Přerušovanou čarou je vyznačena referenční hodnota statického dekodéru (beam pruning pro něj není měněn).

Při snaze posoudit, zda by nebylo vhodné přenastavit hodnotu maximálního počtu živých stavů, byl vytvořen graf 5.14 popisující závislost doby rozpoznávání na maximálním počtu živých stavů a graf 5.15 popisující závislost přesnosti rozpoznávání na maximálním počtu živých stavů. Z těchto grafů lze učinit závěr, že hodnota $MS = 8\,000$ je nastavena korektně.

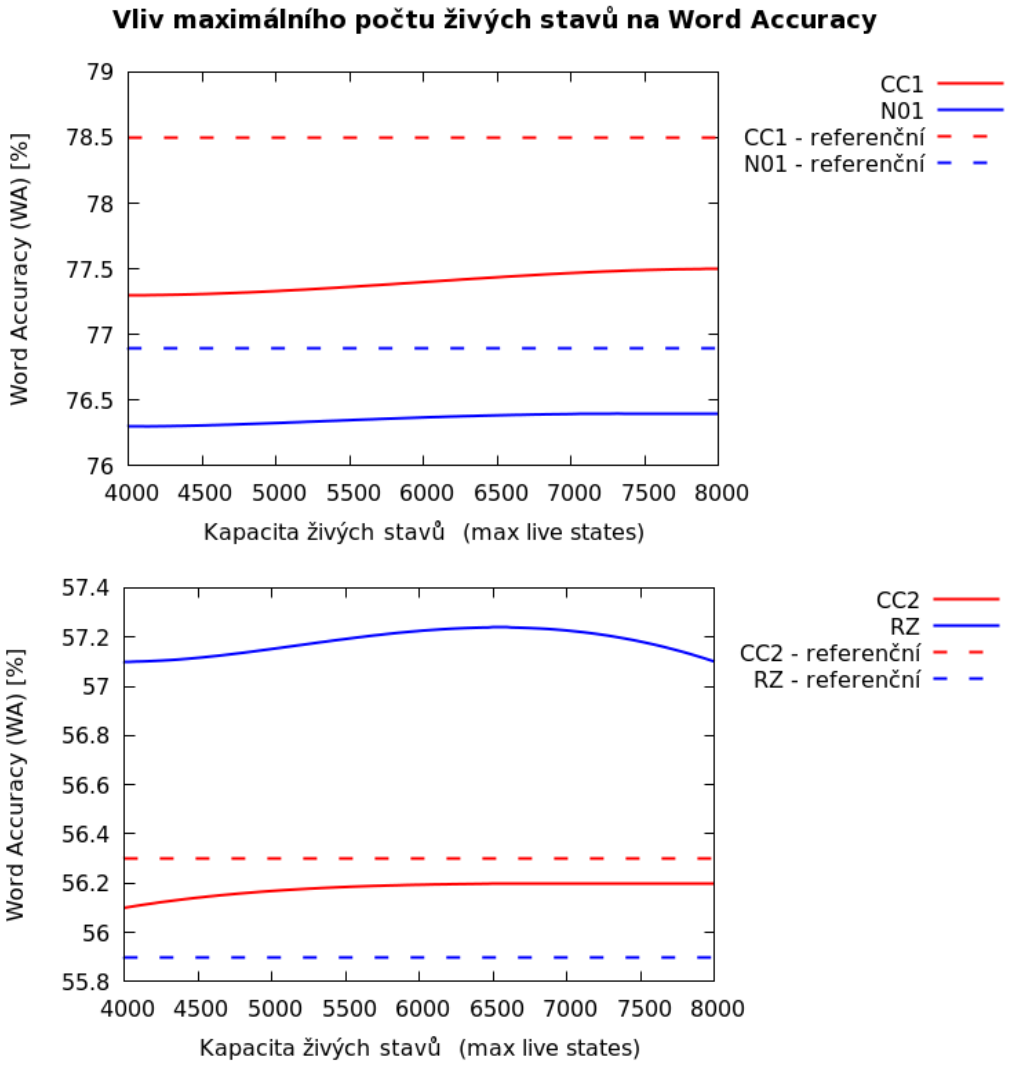


Obrázek 5.14: Graf závislosti doby rozpoznávání jedné nahrávky na maximálním počtu živých stavů. Přerušovanou čarou je vyznačena referenční hodnota statického dekodéru (maximální počet živých stavů pro něj není měněn), za využití podmínek z kapitoly 3.3.

V tabulce 5.3 lze vidět srovnání původního statického dekodéru s dekodérem dynamickým, jehož vhodná konfigurace byla určena na základě experimentů popsaných v této kapitole. Pro výsledné nastavení platí, že $BP = 190$, $MS = 8\,000$ a $K = 15$, kde BP je hodnota Beam Pruningu, MS je maximální počet živých stavů a K odpovídá kapacitě jednotlivých stavů.

	Čas [s]	**Paměť** [KB]	**CC1** (WA)	**CC2** (WA)	**RZ** (WA)	**N01** (WA)
SD	**20.41**	2 028 790	**78.5 %**	56.3 %	55.9 %	**76.9 %**
DD	28.62	**506 112**	77.7 %	**57.0 %**	**56.4 %**	76.4 %

Tabulka 5.3: Porovnání referenčních výsledků (řádek SD) s výsledky dynamického dekodéru (řádek DD), za podmínek z kapitoly 3.3. Beam Pruning pro dynamický dekodér je zde 190, maximální počet živých stavů 8000 a kapacita stavů 15.



Obrázek 5.15: Graf závislosti přesnosti rozpoznávání na maximálním počtu živých stavů. Přerušovanou čarou je vyznačena referenční hodnota, získaná skórováním původního statického dekodéru (maximální počet živých stavů zde není pro statický dekodér měněn), za využití podmínek z kapitoly 3.3.

Kapitola 6

Jazykové modelování pro dynamický dekodér

V kapitole 5 byla popsána výsledná implementace dynamického dekodéru včetně závěrečného porovnání s původním statickým dekodérem (viz tabulka 5.3). Z tabulky výsledků vyplývá, že dynamický dekodér je pro původní jazykový model cca o 30 % pomalejší, avšak vystačí pouze se čtvrtinou operační paměti. Maximální rozdíl v úspěšnosti statického a dynamického dekodéru je 0.8 % dle tabulky 5.3.

Dynamický dekodér nám však přináší nové možnosti při jazykovém modelování, které mají potenciál úspěšnost zvýšit. Při statickém dekódování n-gramovým modelem je nutno vytvořit n-gramovou rozpoznávací síť. N-gramy jazykového modelu pro sestavení této sítě musely být doposud vždy značně prořezány, jelikož by skrze paměťové požadavky nebylo jinak možné výslednou rozpoznávací síť sestavit.

Významnou nevýhodou vytváření nového jazykového modelu a s tím související rozpoznávací sítě pro statický dekodér je také nepredikovatelnost výsledné velikosti takto vzniklé rozpoznávací sítě. Naopak pro dynamický dekodér s implementovanou optimalizací z kapitoly 5.4.2 jsme schopni požadované množství operační paměti pro jazykový model přesně spočítat a hůře predikovatelná je pouze velikost unigramové rozpoznávací sítě, ta se však vždy pohybuje v řádu jednotek MB (pro anglický jazyk s velikostí slovníku okolo 60 000 slov).

Z experimentů v kapitole 5.5 víme, že velikost rozpoznávací sítě statického dekodéru pro trigramový jazykový model o 54 546 unigramech, 675 526 bigramech a 502 833 trigramech vyžaduje cca 1.5 GB operační paměti. Stejný jazykový model je uložen ve vyhledávací struktuře pomocí optimalizace z kapitoly 5.4.2 na pouhých 20 MB. Tento markantní rozdíl v požadované paměti nám umožňuje využívat značně komplexnější jazykové modely. Poznamenejme také, že nejvyšší řád n-gramu jazykového modelu nemá vliv na výslednou paměť při využití optimalizace stromovou strukturou z kapitoly 5.4.2.

Výše uvedená fakta nám vytváří velký prostor pro experimentování s různými komplexnějšími jazykovými modely. Jelikož zdrojová data, ze kterých byl vytvořen původní jazykový model, již nebyla k dispozici, byla vytvořena nová datová sada sestávající z anglických textů získaných prostřednictvím podobných zdrojů jako původní zdrojová data. Tato nová data byla využita pro experimentování s jazykovým modelováním. Jelikož jsou však nově vzniklé jazykové modely sestaveny z jiných dat než model původní, nelze následující výsledky přímo porovnávat s těmi předešlými.

6.1 Tvorba jazykového modelu

V této podkapitole popíšeme postup při tvorbě nového jazykového modelu. Základním stavebním kamenem jsou textová data v požadovaném jazyce. Poznamenejme, že nejlepší jsou data získaná za stejných podmínek, pro které je rozpoznávač vytvářen. Jelikož takových dat je však obvykle nedostatek, protože pro tvorbu jazykového modelu potřebujeme velké množství textů, jsou velmi často použita data z různých rozsáhlých textových zdrojů na internetu (Wikipedia, titulky k filmům, apod.). Nevýhodou takto získaných dat pro jazykový model je skutečnost, že mluvená spontánní řeč se značně liší od psané a zdrojové texty mohou obsahovat slova a slovní spojení z velmi odlišné doménové oblasti oproti oblasti cílové. Uveďme smyšlený příklad rozpoznávače pro klientskou linku pojišťovny a jazykového modelu obsahující všechny texty z Wikipedie v daném jazyce. Jako příklad takovýchto dat pak uveďme následující text: „Žralok bílý (Carcharodon carcharias) či velký bílý žralok, dříve často nazývaný žralok lidožravý, je žralok z čeledi lamnovití vyskytující se většinou v pobřežních vodách.“ viz Wikipedia – Žralok Bílý. Z ukázky je patrné, že jazykový model vzniklý nad velkým množstvím takto irelevantních dat bude pravděpodobně poskytovat zavádějící n-gramové pravděpodobnosti. Častým řešením je pak použití i těchto irelevantnějších zdrojů avšak s nastavením vah dle relevantnosti jednotlivých zdrojů.

Pokud již máme k dispozici dostatečně rozsáhlý textový korpus, následuje tzv. **čištění**. Čištěním lze rozumět proces aplikování sady různých regulárních výrazů, jejichž cílem je převést surová zdrojová data do normalizovaného tvaru. Normalizovaným tvarem rozumějme data obsahující pouze znaky z abecedy daného jazyka, bez interpunkce, s jednotným formátem číselných hodnot, právě jednou mezerou mezi slovy, vhodným rozdělením na promluvy (například po větách) apod.

Nyní máme vyčištěná data z různých zdrojů, ze kterých chceme vypočítat jazykový model. Pro výpočet jazykového modelu byl použit již existující nástroj firmy Phonexia, jenž umožňuje zadat váhy (relevantnost) jednotlivých zdrojů (případně určit tyto váhy automaticky) a pomocí vyhlazovacího algoritmu **Good – Turing** (viz [12]) vypočítat n-gramové pravděpodobnosti jazykového modelu.

Jelikož proces čištění není obvykle stoprocentní úspěšnosti, a i z důvodu minimalizace jazykového modelu, je nutné výsledný jazykový model prořezat (technikou zvanou **pruning**). Pro naše jazykové modelování používáme dvě různé prořezávací techniky.

První prořezávací technika spočívá ve výběru pouze určitého počtu unigramů ze získaného jazykového modelu. Jako příklad uveďme náš neprořezaný jazykový model s 1 705 631 unigramy. Tolika slovy angličtina ani nedisponuje, natož aby tolik slov bylo aktivně používáno v mluvené řeči. Je tedy vhodné stanovit určitý počet nejčastějších slov (nejpravděpodobnějších unigramů) a všechny ostatní z jazykového modelu vymazat, včetně n-gramů vyššího řádu obsahující některá z takto odstraněných slov. V našem případě je ponecháno 60 000 nejpravděpodobnějších slov (tato konstanta byla doporučena zaměstnanci firmy Phonexia zabývajícími se tvorbou jazykových modelů, a také experimentálně ověřena). Aplikací této prořezávací techniky klesl počet n-gramů pětigramového jazykového modelu z 265.4 mil. na 180.7 mil. n-gramů.

Druhá technika, tzv. **entropy-based pruning** viz např. [13], spočívá v odstranění těch n-gramů, jejichž odstranění sníží perplexitu jazykového modelu nejméně. Celkový počet n-gramů je kontrolován prahem. Experimentování s hodnotou prahu (dále ji značme T) a následné zkoumání vlastností vzniklého jazykového modelu je hlavním předmětem této kapitoly.

6.2 Výsledky jazykového modelování

Dále se budeme zabývat provedenými experimenty s jazykovým modelováním. Vycházíme vždy z jazykového modelu prořezaného na 60 000 unigramů, sestaveného z částečně odlišných dat oproti původnímu jazykovému modelu z předchozích kapitol. Experimentováno bylo s různými hodnotami prahu T pro Entropy-based Pruning, různými hodnotami beam pruningu a kapacity stavů. Experimentováno bylo taktéž s řádem jazykového modelu, tedy experimenty lze rozdělit na experimenty s trigramovými a pětigramovými jazykovými modely.

V následující tabulce 6.1 jsou uvedeny jednotlivé hodnoty prahu T a velikosti výsledného jazykového modelu pro konfigurace, které generovaly dobré výsledky.

Práh T	**Počet n-gramů [mil]**				
	1-gramů	**2-gramů**	**3-gramů**	**4-gramů**	**5-gramů**
$\mathbf{7 * 10^{-10}}$	0.060	11.92	21.38	14.88	4.17
$\mathbf{7 * 10^{-9}}$	0.060	4.01	6.24	3.15	0.66
$\mathbf{4 * 10^{-8}}$	0.060	1.33	1.19	0.37	0.05
$\mathbf{10^{-9}}$	0.060	10.00	16.49	0	0
$\mathbf{10^{-8}}$	0.060	2.97	3.38	0	0
$\mathbf{10^{-7}}$	0.060	0.74	0.54	0	0

Tabulka 6.1: Velikost jednotlivých jazykových modelů podle hodnoty prahu T.

U každého modelu řádu N z tabulky 6.1 byla zkoumána rychlost, potřebná operační paměť a úspěšnost na datasetech popsaných v kapitole 3.2. V tabulce 6.2 lze nalézt výsledky těchto experimentů.

T	**N**	**BP**	**K**	**Čas** [s]	**Paměť** [MB]	**CC1**	**CC2**	**RZ**	**N01**
$7 * 10^{-10}$	5	190	15	25.69	1212.48	**73.3**	**52.2**	**58.9**	**79.1**
$7 * 10^{-9}$	5	190	15	27.57	577.65	72.9	51.8	58.8	78.8
$7 * 10^{-9}$	5	175	20	**24.37**	616.48	72.9	51.8	58.6	78.9
$7 * 10^{-9}$	5	175	30	28.43	695.38	73.1	52.0	58.6	**79.1**
$4 * 10^{-8}$	5	190	15	29.25	**454.15**	72.2	51.1	58.4	78.2
10^{-9}	3	190	15	27.15	787.16	**72.5**	**51.8**	**58.7**	**78.4**
10^{-8}	3	190	15	**27.00**	502.69	71.9	51.2	58.2	78.2
10^{-7}	3	190	15	27.48	**428.07**	71.1	50.4	57.8	77.4

Tabulka 6.2: Výčet nejlepších získaných výsledků při experimentování s jazykovým modelováním za podmínek uvedených v kapitole 3.3. Výsledky úspěšnosti jsou reportovány v procentuálním vyjádření word accuracy, pro různé hodnoty beam pruningu (BP) a kapacity jednotlivých stavů (K).

Z tabulky 6.2 můžeme vidět, že 5-gramové jazykové modely poskytují značně lepší výsledky než modely trigramové. Všimněme si, že oproti poslednímu trigramovému jazykovému modelu (s hodnotou prahu $T = 10^{-7}$) poskytuje 5-gramový jazykový model s hodnotou prahu $T = 7 * 10^{-10}$ (viz 6.2) až o 2.2 % lepší word accuracy.

Na provedených experimentech lze také pozorovat, že komplexnější jazykové modely mají pozitivní vliv na časovou náročnost. S vyšší komplexností roste i potřebná operační paměť, avšak poznamenejme, že i u nejkomplexnějšího vyzkoušeného jazykového modelu tvořila požadovaná operační paměť pouze 62 % paměti nutné pro původní statický dekodér.

Kapitola 7

Závěr

V této práci popisujeme základní principy rozpoznávání řeči (kapitola 2) a rozpoznávač implementovaný v BSCORE (Brno Speech Core) firmy Phonexia (viz kapitola 4). Dále se zaměřujeme na problematiku statického dekódování (popsáno v kapitole 2.7), které je v BSCORE rozpoznávači použito, a na způsob modifikace tohoto statického dekodéru v dekodér dynamický (kapitola 5).

Konečným výsledkem této práce je plně funkční a optimalizovaný dynamický dekodér, jehož dynamičnost spočívá v dynamickém aplikování jazykového modelu. Implementovaný dekodér provádí dekódování unigramovou rozpoznávací sítí a při mezislovních přechodech nahrazuje unigramové pravděpodobnosti n-gramovými pravděpodobnostmi, které jsou získány z optimalizované vyhledávací stromové struktury obsahující jazykový model (viz kapitola 5.4.2). Jelikož hledání vhodné podmíněné n-gramové pravděpodobnosti ve vyhledávací struktuře je prováděno velmi často, byla implementována také optimalizace urychlující toto vyhledávání, jež funguje na principu cachování (popsáno v kapitole 5.4.3).

Dynamický přístup k dekódování přináší oproti statickému výhody především z hlediska paměťové úspory a větších možnostech jazykového modelování (viz kapitola 6). Jak je uvedeno v kapitole 5.5, konkrétně v tabulce 5.3, námi implementovaný a optimalizovaný dynamický dekodér vystačí se čtvrtinou operační paměti za cenu o 40 % pomalejšího dekódování. Úspěšnost dekódování je srovnatelná, maximální rozdíl ve word accuracy je mezi statickým a dynamickým dekodérem 0.8 % (viz tabulka 5.3).

Z experimentů v kapitole 6 bylo zjištěno, že úspěšnost dynamického dekodéru lze zvýšit i o 2 % využitím 5-gramového, komplexnějšího jazykového modelu namísto trigramového (komplexnějším zde rozumíme obsahujícím více n-gramových pravděpodobností). Nejkomplexnější 5-gramový jazykový model z kapitoly 6 nejen zvýšil word accuracy oproti více prořezanému trigramovému modelu o průměrně 1.7 %, ale také snížil rozdíl v rychlosti dekódování pouze na 26 % oproti statickému dekodéru. Požadavek na operační paměť pro tento 5-gramový jazykový model je 62 % původně požadované paměti.

7.1 Možnosti pokračování práce

Jak již bylo zmíněno v kapitole 6, dynamický dekodér nám dává výrazně větší prostor k jazykovému modelování. Vzhledem k současným trendům se nabízí otázka zlepšení jazykového modelování nahrazením n-gramového jazykového modelu modelem založeným na rekurentních neuronových sítích (viz např. [9]).

Další výhodou dynamického dekodéru je možnost implementace dynamického přidávání nových slov do rozpoznávače. Pro statický dekodér znamená přidání nových slov do rozpoznávače nutnost překompilovat celou rozpoznávací síť, což je časově poměrně náročný proces. Pro přidání nových slov do našeho dynamického dekodéru stačí překompilovat pouze menší unigramovou rozpoznávací síť a n-gramové pravděpodobnosti nových slov přidat do vyhledávací struktury obsahující jazykový model (pokud nějaké n-gramové pravděpodobnosti známe, v opačném případě jsou přidány jen unigramové pravděpodobnosti). Pro první prototyp dynamického dekodéru či dekodéru využívající optimalizovanou hashovací strukturu z kapitoly 5.4.1 je přidání n-gramů do vyhledávací struktury snadné a spočívá v pouhém vložení pravděpodobností na základě spočteného hashe. Pro dynamický dekodér používající optimalizace z kapitol 5.4.2 a 5.4.3 je přidání n-gramů složitější. Nejefektivnější implementace bychom dosáhli modifikací cache paměti tak, aby obsahovala i permanentní záznamy o nových n-gramových pravděpodobnostech. Nejefektivnějšího přidávání nových slov do rozpoznávače bychom dosáhli implementací plně dynamického dekodéru viz podkapitola 7.2.

7.2 Plně dynamický dekodér

Tato práce se zabývá implementací dekodéru, který dynamicky expanduje jazykový model. Jednotlivá slova jsou však reprezentována staticky v unigramové rozpoznávací síti. Pokud bychom chtěli dosáhnout plně dynamického dekodéru, skládali bychom statickou unigramovou rozpoznávací síť pouze z akustických jednotek a výslovnostního slovníku. Výhodou plně dynamického dekodéru je jednoduchost přidávání nových slov do rozpoznávače, které spočívá v pouhém přidání nových slov do výslovnostního slovníku (pokud máme k dispozici pouze slova bez výslovnosti, výslovnost je možné algoritmicky odhadnout pomocí modulu pro převod grafémů na fonémy). Kromě usnadnění přidávání nových slov do rozpoznávače však plně dynamický dekodér nepřináší žádné další výhody a jeho implementací lze také předpokládat potenciální snížení rychlosti dekodéru.

Literatura

[1] National Institute of Standards and Technology. [Online; navštíveno 01.02.2017].
URL http://www.nist.gov

[2] Sun Grid Engine. [Online; navštíveno 25.02.2017].
URL http://www.fit.vutbr.cz/CVT/cluster/sge.php.cz

[3] Voice biometry and speech techlogoies – Phonexia. [Online; navštíveno 09.12.2016].
URL http://www.phonexia.com

[4] *The 2009 (RT-09) Rich Transcription Meeting Recognition Evaluation Plan*. National Institute of Standards and Technology, 2009.
URL http://www.itl.nist.gov/iad/mig/tests/rt/2009/docs/rt09-meeting-eval-plan-v2.pdf

[5] Fiscus, J. G.; Radde, N.; Garofolo, J. S.; aj.: The Rich Transcription 2005 Spring Meeting Recognition Evaluation. [Online; navštíveno 11.02.2017].
URL http://link.springer.com/chapter/10.1007/11677482_32#page-1

[6] Hinton, G.; Deng, L.; Yu, D.; aj.: *Deep Neural Networks for Acoustic Modeling in Speech Recognition*. Signal Processing Magazine, 2012.

[7] III, J. O. S.: *Spectral Audio Signal Processing*. W3K Publishing, 2011, ISBN 978-0-9745607-3-1.

[8] Jurafsky, D.; Martin, J. H.: *Speech and Language Processing*. Prentice-Hall, 2000, ISBN 978-0131873216.

[9] Mikolov, T.; Karafiát, M.; Burget, L.; aj.: *Recurrent neural network based language model*. Department of Electrical and Computer Engineering, Johns Hopkins University, 2010.

[10] Paul, D.: ngram-format - File format for ARPA backoff N-gram models. [Online; navštíveno 19.02.2017].
URL http://www.speech.sri.com/projects/srilm/manpages/ngram-format.5.html

[11] Povey, D.: Kaldi opensource. [Online; navštíveno 09.12.2016].
URL http://www.kaldi-asr.org

[12] Psutka, J.; Müller, L.; Matoušek, J.; aj.: *Mluvíme s počítačem česky*. ACADEMIA, 2006, ISBN 80-200-1309-1.

[13] Stolcke, A.: Entropy-based Pruning of Backoff Language Models. [Online; navštíveno 08.04.2017].
URL http://www.itl.nist.gov/iad/mig/publications/proceedings/darpa98/html/lm20/lm20.htm

[14] Tilk, O.; Alumäe, T.: *LSTM for Punctuation Restoration in Speech Transcripts*. InterSpeech 2015.

[15] Xiong, W.; Droppo, J.; Huang, X.; aj.: Achieving Human Parity in Conversational Speech Recognition. 2017.

[16] Xu, K.; Xie, L.; Yao, K.: *Investigating LSTM for Punctuation Prediction*. InterSpeech 2013.

[17] Young, S. J.; Russell, N. H.; Thornton, J. H. S.: *Token Passing: a Simple Conceptual Model for Connected Speech Recognition System*. Cambridge University Engineering Department, 1989.

[18] Žmolíková, K.: *Far-Field Speech Recognition*. Diplomová práce, Brno University of Technology, Faculty of Information Technology, Brno, 2016.