

Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Mikołaj Dądela

Nr albumu: 262484

**Corthus – korpus równoległy
z interfejsem internetowym
i przeszukiwarką fonetyczną**

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
dra hab. Adama Przepiórkowskiego
Instytut Informatyki

Wrzesień 2012

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data

Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data

Podpis autora (autorów) pracy

Streszczenie

Praca opisuje tworzenie korpusu równoległego z tekstów liturgicznych w językach polskim, cerkiewnosłowiańskim i starogreckim. Korpus został wyposażony w dwa interfejsy: wiersza poleceń oraz internetowy. Zostało zaimplementowane przeszukiwanie oparte na przybliżonej wymowie słów, aby można było łatwo znaleźć słowo, którego zna się tylko wymowę.

Słowa kluczowe

tłumaczenie, korpus, dopasowanie, alignment, teksty równoległe, wyszukiwanie fonetyczne, metaphone, Python

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

H. Information Systems
H.3 Information Storage and Retrieval
H.3.1 Content Analysis and Indexing

Tytuł pracy w języku angielskim

Corthus – a parallel corpus with internet interface and a phonetic search engine

Spis treści

Wprowadzenie

Cel

Głównym celem mojego projektu było ułatwienie szerszej publiczności dostępu do tekstów liturgicznych w różnych językach i sprawienie, żeby były zrozumiałe dzięki możliwości równoległego ich czytania i łatwo dostępne dzięki wyszukiwarce.

Pragnąłem również ułatwić zainteresowanym naukę języka cerkiewnosłowiańskiego, poczynając już od pierwszego jej etapu, którym jest nauka czytania. Chciałem również poprzez zestawienie tekstów unaocznic analogiczny, a często identyczny szyk zdań w językach starogreckim i cerkiewnosłowiańskim. Trwa obecnie dyskusja na temat zasadności tłumaczenia tekstów liturgicznych na język polski i poprawności istniejących tłumaczeń – chciałem więc stworzyć pomoc do ich badania, jak również do potencjalnego tłumaczenia kolejnych podobnych tekstów na język polski.

Dużym wyzwaniem w realizacji projektu okazało się dopasowywanie tekstów wielojęzycznych, gdyż okazało się, że teksty liturgiczne w zależności od ich redakcji bardzo się różnią, i to sprawia, że dopasowywanie takich tekstów jest trudne. Chciałbym więc również przedstawić algorytmy dopasowywania tekstów użyte w niniejszej pracy. Poświęcam więc temu zagadnieniu osobny rozdział.

Wizja

Planowałem stworzyć serwis internetowy, w którym można łatwo przeglądać, przeszukiwać i porównywać teksty w różnych językach, w pisowni oryginalnej lub w transkrypcji fonetycznej na język polski.

Ważnym elementem projektowanego serwisu była przeszukiwarka wspomnianych tekstów. Głównym jej celem była nie tyle precyzja wyszukiwania, co łatwość jej użycia, gdyż pisownia cerkiewnosłowiańska i starogrecka jest na tyle skomplikowana, że nie można założyć, że użytkownik bezbłędnie wpisze w wyszukiwarkę frazę z szukanego tekstu.

Ciekawą możliwością dalszego rozwinięcia projektu byłaby implementacja rytmu kalendarza prawosławnego i udostępnienie linku „tekst na dzisiaj”.

Mimo, że teksty liturgiczne są głównym przedmiotem zainteresowania niniejszej pracy, to starałem się stworzyć na tyle ogólną implementację, żeby można było jej później użyć do innych zastosowań, np. stworzenia bazy tłumaczeń tekstów technicznych. Niektóre algorytmy (na przykład podział na zdania, transliteracja) w sposób nieunikniony używają różnych wersji dla trzech różnych języków, z których składała się moja baza, ale nic nie stoi na przeszkodzie, żeby dodać analogiczne funkcje do obsługi następnych języków.

Rozdział 1

Algorytmy dopasowania tekstów

Aby zbiór tekstów w różnych językach można było nazwać korpusem równoległym, teksty muszą być do siebie dopasowane. Dopasowanie tekstów równoległych polega na określaniu odpowiadających sobie zdań w poszczególnych językach. Jest to potrzebne do prowadzenia badań lingwistycznych.

Dopasowanie tekstów nie jest jednak zadaniem trywialnym, gdyż podczas tłumaczenia dwa zdania mogą być sklejone w jedno lub jedno może być podzielone na dwa; zdarzają się też opuszczenia i wstawienia nowych zdań. Jest to szczególnie trudne w tekstach liturgicznych, gdyż, w zależności od redakcji, często powtarzające się fragmenty mogą być podane w całości lub skrócone do kilku pierwszych słów; na przykład refren może być podany raz lub też powtórzony po każdej zwrotce pieśni. Zdarzają się też duże fragmenty didaskaliów, które występują tylko w jednej wersji językowej.

Program dopasowujący teksty w niniejszej pracy opiera się na algorytmie, korzystającym z funkcji D szacującej prawdopodobieństwo dopasowania zdań (nazywanej też funkcją podobieństwa), bazując na pamięci tłumaczeniowej TM .

1.1. Algorytm dynamiczny

Jednym z algorytmów, które pozwalają na wygenerowanie dopasowania tekstów, jest algorytm dopasowywania sekwencji opierający się na technice programowania dynamicznego. Jedną z jego wersji, dostosowaną właśnie do dopasowywania zdań można znaleźć w pracy [?]. Algorytm ten i jemu podobne są używane w wielu różnych zastosowaniach, na przykład w bioinformatyce do zestawiania sekwencji genetycznych czy w odnajdywaniu podobnych wzorców w sekwencjach audio i wideo (znany jako Dynamic time warping).

Algorytm ten bazuje na funkcji kosztu, która ocenia podobieństwo dwóch dopasowywanych zdań (duże podobieństwo – niski koszt przyporządkowania). Algorytm w jego najprostszej wersji można zapisać w ten sposób:

```

Input:  $A = a_1, \dots, a_n$ 
Input:  $B = b_1, \dots, b_m$ 
 $cost \leftarrow$  macierz zerowa  $n \times m$  ;
 $cost_{0,0} \leftarrow 0$  ;
foreach  $i$  in  $1 \dots n-1$  do
    foreach  $j$  in  $1 \dots m-1$  do
         $c \leftarrow \infty$  ;
        if  $i > 0 \wedge j > 0$  then
             $c \leftarrow \min(c, cost_{i-1,j-1} - \log D(a_{i-1}, b_{j-1}))$  ;           /* dopasowanie */
        end
        if  $i > 0$  then
             $c \leftarrow \min(c, cost_{i-1,j} - \log D(a_{i-1}, null))$  ;           /* opuszczenie  $a_{i-1}$  */
        end
        if  $j > 0$  then
             $c \leftarrow \min(c, cost_{i,j-1} - \log D(null, b_{j-1}))$  ;           /* opuszczenie  $b_{j-1}$  */
        end
         $cost_{i,j} \leftarrow c$ 
    end
end
return  $cost_{m-1,n-1}$  ;

```

Podana wersja zwraca tylko całkowity koszt dopasowania. Jest to uproszczenie algorytmu, ponieważ przy dopasowywaniu tekstów, zależy nam przede wszystkim na przyporządkowaniu zdań, a całkowity koszt tego przyporządkowania jest mniej istotny. Pomińnię zostały również inne rodzaje “skoków”, potrzebne dla przyporządkowań 2 do 2 lub 1 do 3 zdań. Pomijam jednak te szczegóły na rzecz czytelności kodu – zasada działania algorytmu pozostaje ta sama.

Używaną funkcją kosztu jest funkcja $-\log p_i$, gdzie $p_i = D(s_{i_a}, s_{i_b})$ jest prawdopodobieństwem, że zdanie s_{i_a} jest tłumaczeniem s_{i_b} , ozn. $s_{i_a} \sim s_{i_b}$. Zsumowany koszt dopasowania całego tekstu więc logarytmem iloczynu wspomnianych prawdopodobieństw, czyli estymowanego prawdopodobieństwa poprawnego dopasowania całego tekstu.

1.2. Funkcja podobieństwa D

Kluczowa dla skutecznego działania powyższego algorytmu jest oczywiście dobrze dobrana funkcja podobieństwa $D(s_a, s_b)$.

Funkcja podobieństwa D zastosowana w niniejszej pracy opiera się głównie na parach tłumaczeń. Korzysta ona z pamięci tłumaczeniowej $TM : L \times L \mapsto [0, 1]$. Jest to zbiór par (s_a, s_b) z przyporządkowanymi prawdopodobieństwami $\mathbb{P}(s_a \sim s_b | s_a, s_b)$, gdzie $s_a \sim s_b$ oznacza “ s_a jest tłumaczeniem s_b ”.

TM jest funkcją zdefiniowaną dla wszystkich par napisów, i dla nieznanych par zwraca zawsze małe, niezerowe prawdopodobieństwo. Aby jednak rozróżnić napisy bardziej i mniej prawdopodobne, w przypadku nieznaizienia pary zdań w pamięci TM funkcja D mnoży zwrócone prawdopodobieństwo o własną, heurystyczną ocenę podobieństwa zdań.

Ocena ta opiera się na założeniu (nieprawdziwym), że zawsze jedno słowo w pierwszym zdaniu odpowiada jednemu słowu w drugim zdaniu. Jeśli nie da się słów w ten sposób dopasować, bo zdania różnią się długością, to przyjmujemy, że słowa z któregoś zdania zostały usunięte. Prawdopodobieństwo takiego dopasowania wynosi więc $p_m^{\min(k,l)} * p_r^{|k-l|}$, gdzie p_m jest prawdopodobieństwem dopasowania dwóch słów, a p_r jest prawdopodobieństwem

Rysunek 1.1: Różne wersje pisowni tego samego akapitu

Στίχ. Ἐλέησόν με ὁ Θεὸς κατὰ τὸ μέγα ἔλεός σου καὶ κατὰ τὸ πλῆθος τῶν οἰκτιρμῶν σου, ἐξάλειψον τὸ ἀνόμημά μου.
 Στίχ. Ἐλέησόν με, ὁ Θεός, κατὰ τὸ μέγα ἔλεός σου, καὶ κατὰ τὸ πλῆθος τῶν οἰκτιρμῶν σου ἐξάλειψον τὸ ἀνόμημά μου.
 Στίχ. Ἐλέησόν με, ὁ Θεός κατὰ τὸ μέγα ἔλεός σου καὶ κατὰ τὸ πλῆθος τῶν οἰκτιρμῶν σου ἐξάλειψον τὸ ἀνόμημά μου.
 Στίχ. Ἐλέησόν με ὁ Θεός, κατὰ τὸ μέγα ἔλεός σου, καὶ κατὰ τὸ πλῆθος τῶν οἰκτιρμῶν σου, ἐξάλειψον τὸ ἀνόμημά μου.
 Στίχ. Ἐλέησόν με ὁ Θεὸς κατὰ τὸ μέγα ἔλεός σου καὶ κατὰ τὸ πλῆθος τῶν οἰκτιρμῶν σου ἐξάλειψον τὸ ἀνόμημά μου.
 Στίχ. Ἐλέησόν με ὁ Θεὸς κατὰ τὸ μέγα ἔλεός σου καὶ κατὰ τὸ πλῆθος τῶν οἰκτιρμῶν σου, ἐξάλειψον τὸ ἀνόμημά μου.

usunięcia słowa. W swojej pracy przyjąłem arbitralnie, że $p_m = 0.5, p_r = 0.2$.

Taka ocena odpowiada spostrzeżeniom z pracy [?], że długości odpowiadających sobie zdań są bardzo silnie skorelowane.

1.3. Pamięć tłumaczeniowa TM

Pamięć tłumaczeniowa TM przechowuje często występujące pary tłumaczeń zdań, wraz z ich częstościami występowania i prawdopodobieństwami, że zdania w parze są rzeczywiście swoimi tłumaczeniami.

Aby jednak można było znaleźć często występujące pary tłumaczeń, potrzebne jest dopasowanie tekstów – koło się zamyka. Na szczęście wśród tekstów w bazie są takie, które można dopasować bazując tylko na ich długościach. Teksty te zostały więc dopasowane za pomocą Hunaligna, i początkowa pamięć tłumaczeniowa bazowała właśnie na tym pierwszym dopasowaniu. Zapisywane były tylko pary, które wystąpiły co najmniej 2 razy. Tak dopasowane zdania stanowią dla przeciętnego tekstu polskiego 4.20% całości, 4.18% dla cerkiewnosłowiańskiego i 4.84% dla greckiego.

Kiedy wspomniane pary tłumaczeń są już gotowe, prawdopodobieństwa można obliczyć w następujący sposób:

$$\mathbb{P}(s_1 \sim s_2 | s_1, s_2) = \frac{\mathbb{P}(s_1 \sim s_2, s_1, s_2)}{\mathbb{P}(s_1, s_2)} = \frac{\mathbb{P}(s_1 \sim s_2)}{\mathbb{P}(s_1) + \mathbb{P}(s_2)} = \frac{\#((s_1, s_2))}{\#(s_1) + \#(s_2)},$$

gdzie $\#$ oznacza liczbę wystąpień w korpusie odpowiednio zdania lub pary zdań.

Tak obliczone prawdopodobieństwo jest bardzo sensownym wnioskiem, jednak okazuje się ono niepraktyczne, gdyż pamięć tłumaczeniowa zapytana o parę, w której będzie znala tylko jedno ze zdań, zwróci zerowe prawdopodobieństwo, które spowoduje nieskończony koszt dopasowania. Co gorsza, jeśli oba zdania będą nieznane, podjęta będzie niebezpieczna próba dzielenia przez zero. Aby rozwiązać oba powyższe problemy, stosuje się tzw. wygładzanie¹. W niniejszej pracy zastosowane zostało wygładzanie Laplace’a – wzór przyjmuje wtedy postać:

$$\mathbb{P}(s_1 \sim s_2 | s_1, s_2) = \frac{\#((s_1, s_2)) + \alpha}{\#(s_1) + \#(s_2) + \alpha N},$$

gdzie N to liczba możliwych par utworzonych z dopasowanych zdań, a α jest małą liczbą dodatnią.

Po utworzeniu pamięci tłumaczeniowej okazało się, że w bazie znajduje się wiele różnych pisowni tego samego akapitu (patrz Rys. ??) – aby więc temu zaradzić, wszystkie akapity są przed przetwarzaniem zamienione na ciągi kluczy Metaphone².

¹ang. *smoothing*

²Dokładny opis tego algorytmu można znaleźć w rozdziale ??

1.4. Dopasowanie wstępne

Jak łatwo zauważyć, powyższy algorytm ma złożoność obliczeniową $\mathcal{O}(n^2)$. Aby poprawić jego wydajność, zaimplementowane zostało wstępne dopasowywanie tekstów. Zasada jego działania jest prosta: w obu tekstach wyszukiwane są fragmenty, które występują w parach TM , a następnie dopasowane są sekwencje tych fragmentów. Dzięki temu zamiast dopasowywać od razu cały tekst, można dopasowywać za jednym razem tylko jego małe fragmenty, co znacznie obniża oczekiwany czas działania programu (mimo, że pesymistyczny jest wciąż taki sam). Często to wstępne dopasowanie prowadzi też do poprawienia poprawności dopasowania. Bardzo dobitnie ukazuje to zjawisko rysunek ??.

1.5. Scalanie dopasowań

Aby wyświetlić jednocześnie teksty w trzech językach, potrzebowałem scalić ze sobą dopasowania par tekstów w jedno dopasowanie wszystkich trzech tekstów jednocześnie.

Prostym podejściem do tego zagadnienia jest wybranie z wyrównań tych par zdań, gdzie wszystkie dopasowania są zgodne. Jeśli dopasowania par tekstów są poprawne, w wyniku otrzymamy również poprawne dopasowanie.

W pracy została zastosowana podobna metoda, jednak dla prostoty scalane są tylko 2 dopasowania: polskiego z cerkiewnosłowiańskim i cerkiewnosłowiańskiego z greckim.

1.6. Ewaluacja dopasowań

Ewaluacja dopasowania staje się łatwa, jeśli zamienimy dopasowanie na zbiór par z wagami – wówczas można ocenić dopasowanie używając miar dokładności i pełności³, szeroko stosowanych w systemach uczenia maszynowego.

Zamiana dopasowania na zbiór (oznaczony dla porządku S) przebiega następująco:

Input: A – dopasowanie tekstów w formie par zbiorów indeksów

$S \leftarrow \{\}$;

$W \leftarrow \text{map} : \mathbb{N} \times \mathbb{N} \mapsto [0, 1]$;

foreach (I, J) – para zbiorów indeksów z dopasowania A **do**

if $I = \emptyset$ **then** $I \leftarrow \{\text{null}\}$;

if $J = \emptyset$ **then** $J \leftarrow \{\text{null}\}$;

$S = S \cup (I \times J)$;

foreach (i, j) **in** $I \times J$ **do**

$| W[i, j] \leftarrow |I| \cdot |J|$

end

end

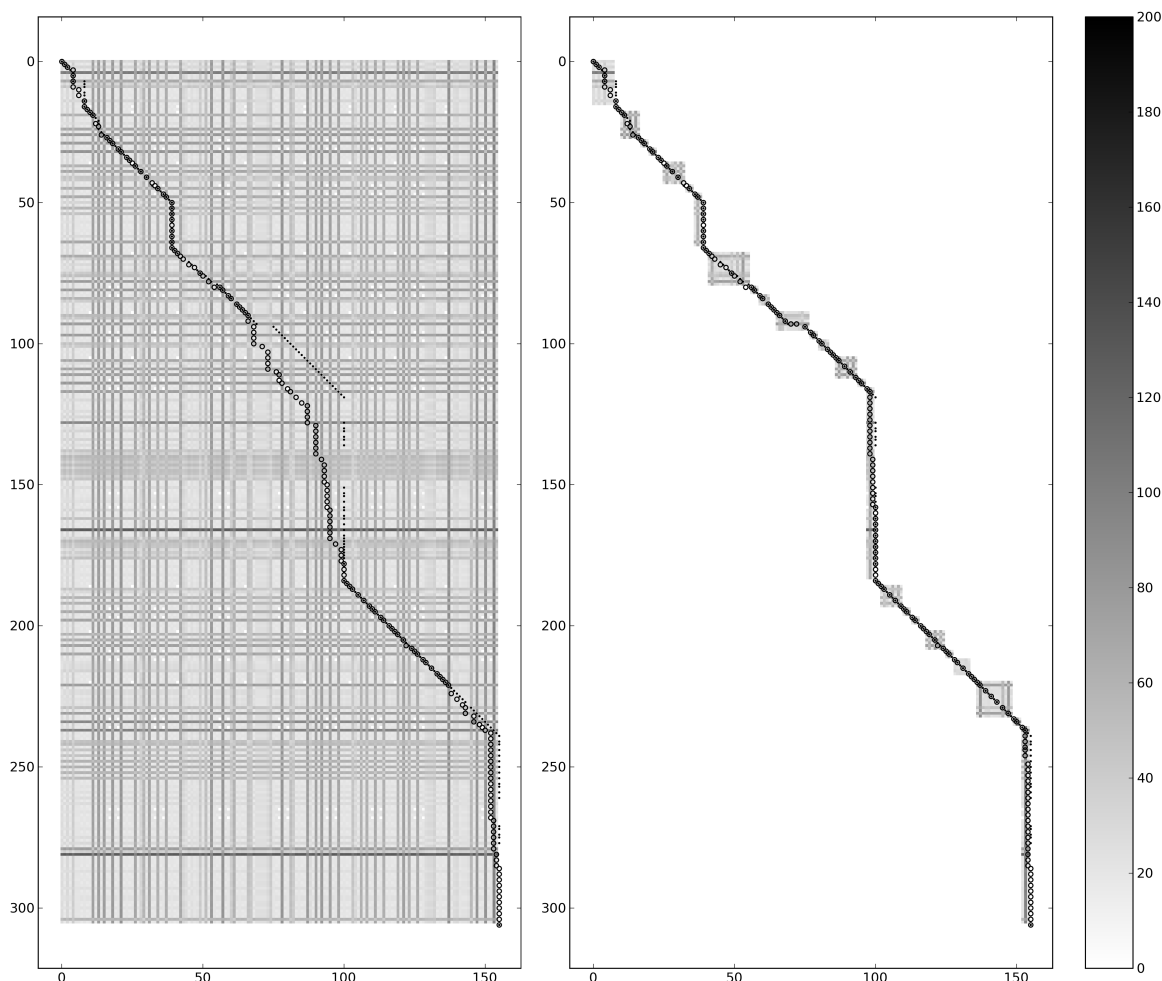
return S

Następnie, kiedy uzyskaliśmy już zbiory par S_1 , S_2 i wagi W_1 , W_2 dla odpowiednio wygenerowanego i wzorcowego dopasowania, możemy obliczyć dokładność i pełność dopasowania:

$$\text{precision} = \frac{\sum_{(i,j) \in S_1 \cap S_2} W_1[i, j]}{|S_1|}, \quad \text{recall} = \frac{\sum_{(i,j) \in S_1 \cap S_2} W_2[i, j]}{|S_2|}$$

³ang. *precision* i *recall*

Rysunek 1.2: Macierz kosztów.



Macierz kosztów dla dopasowania poszczególnych zdań tekstu `kanon_izr`, w językach polskim i cerkiewno-słowiańskim. Oś X odpowiada zdaniom tekstu polskiego, oś Y – zdaniom tekstu cerkiewno-słowiańskiego. Kolor piksela (i, j) obrazka odzwierciedla koszt dopasowania zdania a_i do b_j (nie jest to więc zsumowany koszt, przechowywany w tablicy ‘cost’ algorytmu).

Dwa przedstawione wykresy ukazują dwa tryby działania algorytmu – po lewej widoczna jest pełna macierz kosztów, którą oblicza algorytm w podstawowej wersji. Po prawej ukazany jest wynik działania programu, w wersji ze wstępnym dopasowaniem dobrze pasujących fraz, i odpowiednie fragmenty macierzy kosztów, które są wtedy obliczane.

Ścieżka złożona z kółek biegnąca przez wykres oznacza wygenerowane dopasowanie tekstów – każdy jej punkt to przyporządkowanie sobie zdań. Ścieżka złożona z punktów oznacza poprawne, wzorcowe dopasowanie. Ścieżki nie są ciągłe w miejscach, gdzie mamy do czynienia z dopasowaniem wielu zdań (notabene, w przypadku takich dopasowań koszt jest liczony dla połączonych zdań, więc inaczej, niż ukazuje wykres).

Ciemniejsze poziome i pionowe paski odpowiadają dłuższym zdaniom, ponieważ koszt dopasowania zdań jest bardzo silnie skorelowany z ich długością. Pary zdań, które występują w *TM* nie wyróżniają się na tym wykresie (mają niemal biały kolor, który łatwo się gubi w jasnoszarym otoczeniu).

Wyniki dla lewego wykresu: Precision: 67.30%, Recall: 66.73%

Wyniki dla prawego wykresu: Precision: 81.71%, Recall: 89.22%

Rozdział 2

Opis projektu

Swoją pracę nad projektem zacząłem od pobrania na swój komputer tekstów źródłowych.

2.1. Konwersja danych wejściowych

Teksty pobrałem ze stron [?], [?] i [?]. Pliki z tych stron były odpowiednio w formatach HIP¹, HTML i DOC.

Zdecydowałem wszystkie te pliki przekonwertować do zwykłego tekstu kodowanego w UTF-8. Napisałem w tym celu skrypt w Pythonie, który korzystał dodatkowo z Elinksa i AbiWorda. Zdecydowałem się na kodowanie UTF-8, bo dzięki niemu mogłem używać jednego kodowania dla wszystkich plików, i uznałem, że to uproszczenie rekompensuje 2 razy większy rozmiar pliku. Decyzja o przejściu na zwykły tekst oznaczała również rezygnację z formatowania. Połamałem akapity na linie nie dłuższe niż 70 znaków, rozdzielając je pustymi liniami (tak jak w TeXu), tak żeby można było je wygodnie przeglądać w dowolnym edytorze tekstu lub w konsoli.

2.2. Eksperymenty, próby i błędy

Zacząłem swoją pracę od poszukiwań gotowych narzędzi, które odpowiadałoby przynajmniej części założeń opisanych powyżej. Postaram się tu krótko opisać swoje doświadczenia z nimi.

Hunalign Hunalign jest narzędziem zestawiającym 2 teksty na poziomie zdań (zakładając, że są już podzielone na zdania), bazując na ich długościach i (opcjonalnie) na słowniku. Jeśli jednak nie dysponujemy słownikiem, można uruchomić Hunaligna z opcją `-realign` – Hunalign wówczas po pierwszym, wstępnym dopasowaniu próbuje wygenerować przybliżony słownik, a następnie dopasowuje teksty ponownie z użyciem tego słownika. Dopasowania generowane przez Hunaligna są niemal w 100% poprawne, jeśli teksty dokładnie sobie odpowiadają. Niestety teksty, z którymi pracowałem, bardzo często różniły się od siebie dużymi fragmentami tekstu, co powodowało przesunięcia w dopasowaniu.

Mimo to Hunalign okazał się bardzo przydatny, gdyż, bazując głównie na długości zdań, generuje dopasowanie, które może zostać wykorzystane do zapełnienia pamięci tłumaczeniowej, która następnie może posłużyć do polepszenia dopasowania.

¹HIP – standard zapisu tekstu cerkiewnosłowiańskiego. Więcej szczegółów w dodatku ??

GIZA++ GIZA++ [?] jest narzędziem tworzącym dopasowanie na poziomie słów z już dopasowanych par zdań. Nie użyłem go w mojej pracy, ponieważ przede wszystkim potrzebne było dopasowanie na poziomie zdań. Dopasowanie na poziomie słów byłoby niewątpliwie ciekawym i użytecznym rozszerzeniem tej pracy, jest jednak poza jej zakresem.

Poliqarp Poliarp [?] jest wolnym (i darmowym) zestawem narzędzi do przeszukiwania dużych korpusów. Wspiera obsługę korpusów otagowanych znacznikami morfosyntaktycznymi i pozwala na przeszukiwanie korpusu za pomocą złożonych zapytań. W początkowej fazie projektu była planowana integracja niniejszego projektu z Poliarpem, jednak teksty, którymi się zajmowałem w swojej pracy, nie były otagowane, więc nie mogłbym też wykorzystać pełni możliwości Poliarpa. Poliarp nie posiada również obsługi korpusów równoległych.

Wzorce odmiany i lematyzacja Próbowałem uprościć Hunalignowi generowanie przybliżonego słownika poprzez usunięcie końcówek morfosyntaktycznych ze słów. Nie znalazłem jednak gotowej biblioteki, wykonującej takie zadanie. Rozważałem ręczną implementację tej funkcjonalności, ale po przyjrzeniu się wzorcom odmiany języka cerkiewnosłowiańskiego [?] zrezygnowałem. Początkowo przybliżałem lematyzację skracaniem słowa do 5 liter, a po implementacji Metaphone zauważyłem, że Metaphone również łączy pod wspólnym kluczem wiele form fleksyjnych jednego słowa, więc w końcowej wersji projektu dopasowuję do siebie nie oryginalne teksty, lecz ciągi kluczy Metaphone z nich utworzone.

Słowniki Szukałem również słownika cerkiewnosłowiańsko-polskiego, lecz znalazłem tylko jeden [?], autorstwa O. Aleksego Znosko, i nie udało mi się go zdobyć w wersji cyfrowej – na mail do wydawcy otrzymałem odpowiedź, że autor zmarł przed wydaniem tego słownika, i plik z zawartością słownika nigdzie się nie zachował.

2.3. Architektura projektu

Cała implementacja została napisana w języku Python.

Główna funkcjonalność została umieszczona w katalogu **toolkit**. Wiele skryptów z tego folderu można wywołać z linii komend, aby wykonać różne zadania (patrz rozdział ??).

Interfejs internetowy znajduje się w folderze **www**. Został on szczegółowo opisany w rozdziale ??.

Katalog **texts** zawiera wszystkie teksty w postaci plików tekstowych, i odpowiadające im pliki dopasowań. Moja decyzja o używaniu plików tekstowych zamiast bazy danych była powodowana prostotą operacji na plikach w konsoli, i możliwością łatwej zmiany w organizacji tekstów. Łatwość zmian była dla mnie bardzo ważna, bo w początkowej fazie rozwoju projektu nie było jasne, w jaki sposób system będzie korzystał z tekstów.

Nazwa pliku z tekstem źródłowym ma postać **texts/książka/rozdział/język.txt**, gdzie *język* jest jednym spośród “pl”, “cu”² i “el”.

Katalog **external** zawiera zewnętrzne biblioteki i narzędzia, z których korzysta projekt. Są to Hunalign [?] i Whoosh [?].

²“cu” oznacza język cerkiewnosłowiański, zgodnie ze standardem ISO 639-1.

Tabela 2.1: Uproszczona tablica podstawień dla wersji algorytmu Metaphone zastosowanej w projekcie

metaphone	pl	cu	el
a	a	ʌ	α
e	e, ę	ɛ, ɤ, ɸ	ε, αι
i	i, y	ɪ, ɨ, ɤ, ɥ, ɩ	ι, η, υ
o	o, ą	ɔ, ɒ	ο, ω
u	u, ó	ʊ, ʉ	ου
b	b	β	μπ
c	c, ć, cz	ɕ, ɥ, ɸ	τσ
d	d, dz, dź	ɗ	δ
f	f	ɸ, ɸ	φ, θ
h	g, h, ch	ɣ, ɣ	γ, χ
j	j	ɨ	
k	k	κ	κ
l	l	λ	λ
7	ł	ʌ	
m	m	μ	μ
n	n, ń	ɲ	ν
p	p	π	π
r	r	ρ	ρ
s	s	ɛ	σ, ς
t	t	ɾ	τ
v	w	β, ɱ	β, υ
z	z, ź	ʒ, ʒ	ζ
2	ż, rz, sz, ś	ʒ, ɬ	

Tabela 2.2: Przykładowe klucze Metaphone i przykłady słów im przyporządkowanych

2c2lv	szczęśliwe szczęśliwy szczęśliwa
2l	szli żalu żyli ślę żal żale
7k	łk łuku łuk λδκκ λδκκ λąkę
avvkm	άββακδμκ άββακούμ
b2j	ββίλλ ββίλλο βββίλλ bożej
d7h	długi długo dziełach δόλγα δόλγη
dbrvln	dobrowolnej dobrowolną dobrowolnymi
imnnts	ύμνούντας ύμνούντάς ύμνούντές
hrnj	górnej górniń górniń gorńińλ
isclv2	ήεψλββββββββ ήεψλββββββ
j	ja ń j λ ń je ja
km	κώμη κομβ κίημα κημῶ κόημα κῶμα
lhsms	λογισμούς λογισμούς λογισμοίς
ltrh	λῑτῑργίη λειτουργία liturgia
mh7	mogły mgłę μιλω mgłą μίχαιλδ
mtr	μητέρα μητρί μῑτρη μέτρον μῑτρε
o	δλ ῶ ῶ ῶ οῖα οῖου ο ο ο δ δ
ohnnnj	δγννννλλ δγννννλλ δγννννλλ
pcc	począć pieczęci pieczęcie pieczęć
phd	pochodzi ποχολλ pochodzą pogode
svc	εββββδ εββββδ εββββδ siewcy
t	τλ τό τδ τλ τω τῑ τῑ του τῑ τῑν
t2	téz toś tuż toż tyrze trzy τότρε
zvr	εβββββε zawrę εββββε zawory

2.7. Interfejs wiersza poleceń

Większość modułów pakietu `toolkit` udostępnia swoją funkcjonalność poprzez prosty interfejs linii komend. Większość skryptów przy uruchomieniu bez argumentów wyświetla pomoc lub inne pożyteczne informacje. Wymienię poniżej kilka istotniejszych z nich:

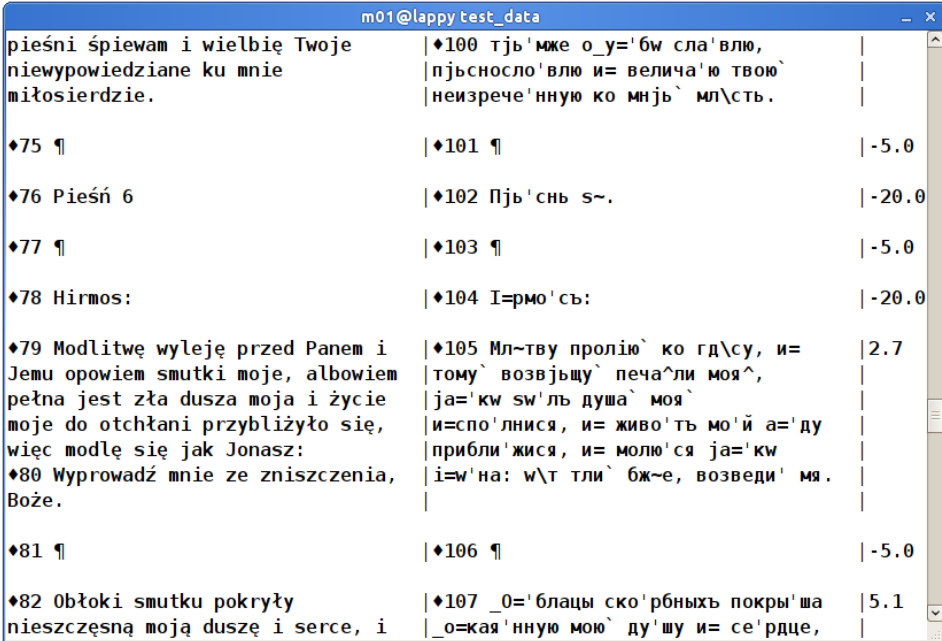
fetcher Fetcher pobiera odpowiedni tekst lub dopasowanie tekstów i wypisuje go (w ładnie sformatowanej postaci) na standardowe wyjście.

search Skrypt przeszukuje bazę tekstów i wyświetla wynik. Wywołanie go z opcją `--create-index` powoduje utworzenie indeksu potrzebnego do realizacji wyszukiwania. Wyszukiwarka opiera się na bibliotece Whoosh [?].

aligner Dopasowuje do siebie 2 teksty. Na uwagę zasługują opcje `--hand`, `--prealign` i `--plot`. Opcja `--hand` pozwala wymusić dopasowanie pewnych par zdań, tak, żeby “nakierować” program na właściwą ścieżkę dopasowania. Opcja `--prealign` działa podobnie, lecz tutaj te pary zdań wyszukiwane są automatycznie. Wreszcie opcje `--plot` i `--plot-sim` pozwalają na wygenerowanie wykresu podobnego do tego przedstawionego na rys. ??.

Alignment Skrypt `Alignment.py` zawiera klasę obsługującą dopasowania wielu tekstów. Wywołany samodzielnie z wiersza poleceń, wyświetla podany plik dopasowania. Fragment przykładowego wyniku działania skryptu można zobaczyć na rysunku ??.

Rysunek 2.1: Przykładowe dopasowanie tekstów wyświetlone w konsoli.



m01@lappy test_data		
pieśni śpiewam i wielbię Twoje	♦100 тѣ'мже о_у='бѡ сла'влю,	
niewypowiedziane ku mnie	пѣсносло'влю и= велича'ю твою`	
miłosierdzie.	неизрече'нную ко мнѣ' мл'сть.	
♦75 ¶	♦101 ¶	-5.0
♦76 Pieśń 6	♦102 Пѣ'снь s~.	-20.0
♦77 ¶	♦103 ¶	-5.0
♦78 Hirmos:	♦104 I=рмо'съ:	-20.0
♦79 Modlitwę wyleję przed Panem i	♦105 Мл~тву пролію` ко гд\су, и=	2.7
Jemu opowiem smutki moje, albowiem	тому` возвѣщу` печа^ли моя^,	
pełna jest zła dusza moja i życie	ја='кѡ sw'ль душа` моя`	
moje do otchłani przybliżyło się,	и=спо'лнися, и= живо'ть мо'й а='ду	
więc modlę się jak Jonasz:	прибли'жися, и= молю'ся ја='кѡ	
♦80 Wyprowadź mnie ze zniszczenia,	i=w'на: w\т тли` бж~е, возведи' мя.	
Boże.		
♦81 ¶	♦106 ¶	-5.0
♦82 Obłoki smutku pokryły	♦107 _o='блaцы ско'рбныхъ покры'ша	5.1
nieszczęsną moją duszę i serce, i	_o='кая'нную мою` ду'шу и= се'рдце,	

Pakiet translit Wszystkie skrypty z pakietu `translit` można uruchomić z linii poleceń, aby dokonać odpowiednią transliterację na wybranym tekście. Jako argument należy podać plik z tekstem, lub `-`, aby korzystać ze standardowego wejścia. Dostępne są następujące skrypty:

- `expand_cu.py` – rozwijanie skrótów języka cerkiewnosłowiańskiego i konwersja liczb na cyfry arabskie,
- `cu2pl.py` – transkrypcja fonetyczna języka cerkiewnosłowiańskiego na polski,
- `render_cu.py` – konwersja tekstu cerkiewnosłowiańskiego do formatu UCS (patrz do-datek ??),
- `simplify_el.py` – ujednolicanie greckich akcentów i usuwanie innych znaków diakrytycznych,
- `el2pl.py` – transkrypcja fonetyczna z (“uproszczonego”) greckiego na polski,
- `metaphone.py` – zamiana tekstu na ciąg kluczy Metaphone.

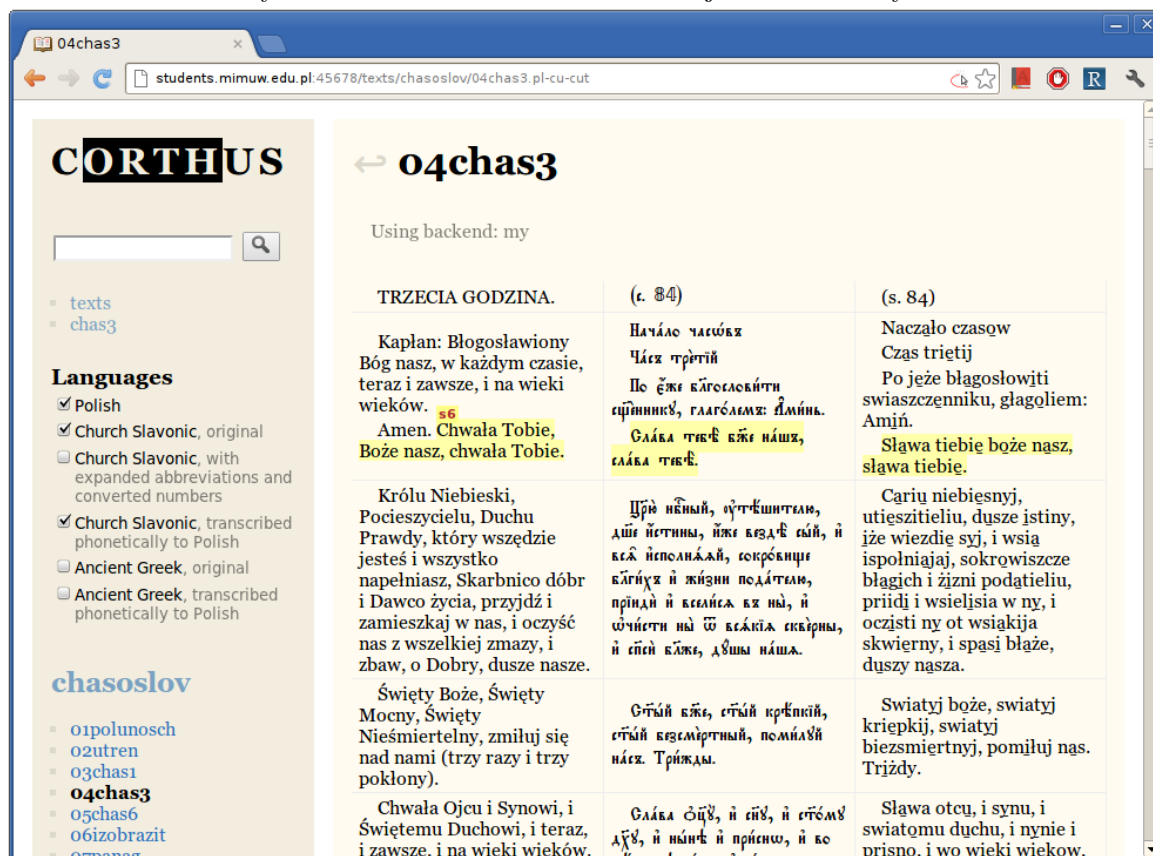
2.8. Interfejs internetowy

Do utworzenia interfejsu internetowego użyłem frameworka Django [?]. Zdecydowałem się na ten wybór, ponieważ cały kod mojego projektu został napisany w Pythonie, a Django jest najpopularniejszym frameworkiem do tworzenia aplikacji internetowych w Pythonie, i miałem z nim styczność podczas zajęć na MIM UW.

Interfejs internetowy można uruchomić do przetestowania wpisując polecenie `./www/manage.py runserver` w terminalu w głównym katalogu projektu. Powoduje to uruchomienie serwera HTTP, który odpowiada na zapytania pod adresem `http://127.0.0.1:8000/`.

Wygląd przykładowej strony można zobaczyć na rys. ??.

Rysunek 2.2: Widok tekstu w interfejsie internetowym.



Model Model danych i operacje na nich obsługuje napisany przeze mnie pakiet `toolkit`. Nie potrzebowiałem używać baz danych, które zwykle są pomocne w podobnych projektach.

Widoki Cały serwis internetowy obsługują 3 widoki.

- **Widok folderu** – Wyświetla spis treści książki lub spis książek. To zadanie sprowadza się do wyświetlenia odpowiednio sformatowanej listy plików.
- **Widok tekstu** – Wyświetla tekst równoległy. Po lewej stronie znajdują się pola wyboru języków, których zmiana powoduje przeładowanie strony z nowymi ustawieniami.
- **Widok wyników wyszukiwania**

Szablony Szablony są prostymi plikami HTML, korzystającymi z zewnętrznych arkuszy stylów CSS i skryptów w JavaScriptcie. Aby umożliwić przeglądanie tekstów cerkiewnosłowiańskich osobom, które nie mają zainstalowanej w systemie odpowiedniej czcionki, użyłem deklaracji `@font-face` z języka opisu formy prezentacji stron WWW CSS. Deklaracja ta wprowadzona została dopiero w wersji 3 standardu, dlatego może nie działać w starszych przeglądarkach. Jednak nowe wersje wszystkich wiodących przeglądarek (Seamonkey 2.7.2, Opera 10.60, Firefox 3.6, Safari 4.0, Internet Explorer 9) obsługują ją poprawnie. Starsze wersje Internet Explorera wyświetlają odstęp pod znakami diakrytycznymi, jednak zawsze można ten problem obejść, wyświetlając tekst w transkrypcji fonetycznej.

Skrypty Zmiana języków jest obsługiwana przez krótki skrypt w języku JavaScript. Preferencje użytkownika są zapisywane w pliku cookie, i odczytywane przy następnej wizycie. Podświetlanie odpowiadających sobie fragmentów tekstu jest również obsługiwane przez skrypt JavaScript.

Dodatek A

Formaty plików

A.1. HIP i UCS – tekst cerkiewnosłowiański

W czasach, kiedy większość tekstów cerkiewnosłowiańskich została zdigitalizowanych, standard Unicode nie zawierał wszystkich znaków potrzebnych do zapisu tekstu w języku cerkiewnosłowiańskim. W odpowiedzi na to powstał standard HIP (opisany dokładnie w internecie na stronie [?]). Pliki HIP to pliki tekstowe kodowane w Windows-1251, używające cyrylicy wraz z niektórymi literami łacińskimi, aby w ten sposób wyrazić wszystkie litery alfabetu cerkiewnosłowiańskiego. Zapis poszczególnych liter alfabetu (różne warianty zapisu) zostały przedstawione w tabeli ??.

Brakujące znaki były stopniowo dodawane do Unicode w wersjach 5.0.0 (2006), 5.1.0 (2008) i 5.2.0 (2009). Można więc obecnie zapisać tekst cerkiewnosłowiański w Unicode – głównym problemem jest jednak dostępność czcionek: czcionki obsługujące tytuła literowe (znaki 2DE0–2DFF) i literę Ѹ (A64B) są wielką rzadkością. Na chwilę obecną udało mi się w internecie znaleźć tylko jedną czcionkę obsługującą w pełni język cerkiewnosłowiański, oznaczoną jako “wersja beta” [?]. Używam wobec tego starszych czcionek, dostosowanych do kodowania UCS. Kodowanie UCS jest oparte na Windows-1251. Znaki cyrylicy są w nim kodowane tak samo, jednak wszystkie pozostałe pozycje strony kodowej są zajęte przez pozostałe znaki języka cerkiewnosłowiańskiego oraz ligatury tych znaków z tytułami lub znakami diakrytycznymi (tylko w zestawieniach, gdzie takie ligatury są potrzebne).

Dużą wadą takiego kodowania jest to, że jest ono niekompatybilne z ASCII, i tak zakodowany tekst jest czytelny tylko jeśli zostanie wyświetlony odpowiednią czcionką. Przykładowe czcionki dla tego kodowania można pobrać ze strony *Irmologion* [?].

A.2. TXT – pliki tekstowe

Wszystkie pliki TXT w projekcie są kodowane w UTF-8. Zdecydowałem jednak nie konwertować tekstów cerkiewnosłowiańskich z zapisu HIP na znaki odpowiadające im według standardu Unicode, ze względu na problemy z wyświetlaniem tych znaków (patrz sekcja ??). Ta decyzja pozwoliła mi na duże uproszczenie logiki programu, bo mogłem bardzo łatwo przeglądać te pliki i wprowadzać w nich zmiany. Wadami tego rozwiązania była konieczność rezygnacji z formatowania i nieco większy rozmiar plików.

Połamam akapity na linie nie dłuższe niż 70 znaków, rozdzielając akapity od siebie pustymi liniami (analogicznie do TeXa), tak żeby można było je wygodnie przeglądać w dowolnym edytorze tekstu lub w konsoli.

Tabela A.1: Kodowanie znaków cerkiewnosłowiańskich w standardzie HIP

A a.....	Ɑ Ɑ	С с.....	Ѐ Ɽ
Б б.....	Ɱ Ɱ	Т т.....	ⱥ ⱥ
В в.....	Ɒ Ɒ	У у.....	ⱦ ⱦ
Г г.....	Ⱳ Ⱳ	_у _у <у> <у>.....	Ⱨ Ⱨ Ⱨ Ⱨ
Д д.....	ⱬ ⱬ	О_у О_у о_у О<у> О<у> о<у>	ⱨ ⱨ ⱨ ⱨ ⱨ ⱨ
Е е.....	ⱪ ⱪ	Ф ф.....	Ⱪ Ⱪ
_Е _е <Е> <е>.....	ⱪ ⱪ ⱪ ⱪ	Х х.....	ⱪ ⱪ
Ж ж.....	ⱬ ⱬ	W\т w\т.....	Ⱬ Ⱬ
З з.....	Ɱ Ɱ	Ц ц.....	ⱬ ⱬ
И и.....	Ɒ Ɒ	Ч ч.....	Ɑ Ɑ
I i.....	Ⱳ Ⱳ	Ш ш.....	Ɱ Ɱ
_i <i>.....	Ⱳ Ⱳ	Щ щ.....	Ɐ Ɐ
Й й.....	ⱬ ⱬ	Ъ ъ.....	Ɒ Ɒ
К к.....	Ɱ Ɱ	Ы ы.....	ⱱ ⱱ
Л л.....	ⱬ ⱬ	Ь ь.....	ⱱ ⱱ
М м.....	Ɱ Ɱ	Ј ѣ Ј ѣ ј ѣ.....	Ⱳ Ⱳ Ⱳ
Н н.....	Ɒ Ɒ	Ю ю.....	ⱳ ⱳ
О о.....	Ⱳ Ⱳ	ЈА Ја ја.....	ⱴ ⱴ ⱴ
_О _о <О> <о>.....	Ⱳ Ⱳ Ⱳ Ⱳ	Я я.....	Ⱶ Ⱶ
W w.....	ⱬ w	_Кс _Кс _кс <Кс> <Кс> <кс>	ⱶ ⱶ ⱶ ⱶ ⱶ ⱶ
_W _w <W> <w>.....	ⱬ ⱬ ⱬ ⱬ	_Пс _Пс _пс <Пс> <Пс> <пс>	ⱷ ⱷ ⱷ ⱷ ⱷ ⱷ
П п.....	Ɒ Ɒ	F f.....	ⱸ ⱸ
Р р.....	Ⱳ Ⱳ	V v.....	ⱹ ⱹ

A.3. Pliki dopasowań

Dopasowania tekstów są zapisywane w formacie CSV, z tabulatorem pełniącym rolę separatora pól. Jest to dokładnie taki format, jaki zwraca Hunalign.

Nazwy plików dopasowań kończą się rozszerzeniem `.hunalign`, `.my` lub `.golden`, w zależności od swojego pochodzenia. Kolejne rozszerzenia oznaczają odpowiednio pliki wygenerowane przez Hunaligna, pliki utworzone przez aligner z projektu i pliki dopasowań utworzone ręcznie.

Bibliografia

- [1] Kenneth W. Church, William A. Gale, *A Program for Aligning Sentences in Bilingual Corpora*, Association of Computational Linguistics, 1993
- [2] Teksty cerkiewnosłowiańskie:
<http://orthlib.ru/worship/>
- [3] Teksty starogreckie:
<http://analogion.gr/glt/>
- [4] Teksty polskie:
<http://www.liturgia.cerkiew.pl/page.php?id=14>
- [5] Opis standardu HIP (w jęz. rosyjskim):
<http://orthlib.ru/hip/hip-9.html>
- [6] Projekt Ponomar
http://www.ponomar.net/cu_support.html
- [7] Czcionki z ligaturami potrzebne do wyświetlania tekstów w jęz. cerkiewnosłowiańskim:
<http://www.irmologion.ru/fonts.html>
- [8] The Unicode Consortium. *The Unicode Standard*.
<http://www.unicode.org/charts/PDF/U0400.pdf> (Cyrillic)
<http://www.unicode.org/charts/PDF/U2DE0.pdf> (Cyrillic Extended-A)
<http://www.unicode.org/charts/PDF/UA640.pdf> (Cyrillic Extended-B)
- [9] Algorytm Soundex:
<http://en.wikipedia.org/wiki/Soundex>
- [10] Algorytm Metaphone:
<http://en.wikipedia.org/wiki/Metaphone>,
Hanging on the Metaphone, Lawrence Philips. Computer Language, Vol. 7, No. 12 (December), 1990.
- [11] GIZA++
<http://code.google.com/p/giza-pp/>
Franz Josef Och, Hermann Ney. "A Systematic Comparison of Various Statistical Alignment Models",
Computational Linguistics, volume 29, number 1, pp. 19-51 March 2003.
- [12] Hunalign
<http://mokk.bme.hu/resources/hunalign/>
D. Varga, L. Németh, P. Halácsy, A. Kornai, V. Trón, V. Nagy (2005). *Parallel corpora for medium density languages*, In Proceedings of the RANLP 2005, pages 590-596.

- [13] Poliqarp
<http://poliqarp.sourceforge.net/about.html>
Adam Przepiórkowski. (2004). *Korpus IPI PAN. Wersja wstępna*. IPI PAN, Warszawa.
- [14] Stanisław Strach, *Krótką gramatyka języka cerkiewnosłowiańskiego*. Prawosławna Diecezja Białostocko-Gdańska, 1994.
- [15] Aleksy Znosko, *Słownik cerkiewnosłowiańsko-polski*, Prawosławna Diecezja Białostocko-Gdańska, 1996.
- [16] Django core team (2011). *Django: A Web framework for the Python programming language*. Django Software Foundation, Lawrence, Kansas, U.S.A.
<http://www.djangoproject.com>
- [17] Whoosh Python Search Library
<https://bitbucket.org/mchaput/whoosh/wiki/Home>