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 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 Retreival
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 unaocznić 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 teksty liturgiczne w zależności od ich redakcji bardzo się różnią. Chciałbym więc również przedstawić algorytmy dopasowywania tekstów użyte w niniejszej pracy. Poświęcam 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 występują w różnych wersjach dla trzech różnych języków, z których składał się korpus, ale nic nie stoi na przeszkodzie, żeby dodać analogiczne funkcje do obsługi następnych języków.

Zarys pracy

W pracy zostało zaimplementowane automatyczne dopasowywanie tekstów, z możliwością łatwego ręcznego wprowadzania poprawek. Algorytmy używane do dopasowywania tekstów zostały opisane w rozdziale 1.

System został wyposażony w interfejs internetowy, w którym użytkownik może otworzyć wybrany tekst, jak również przeszukiwać bazę tekstów używając przybliżonej wymowy słów, a nie ich dokładnej pisowni. Dopasowanie tekstów zostało unaocznione poprzez złożenie tekstów w jedną tabelę i dynamiczne podświetlanie odpowiadających sobie elementów tekstu. Została również zaimplementowana transkrypcja fonetyczna z języków obcych na polski, aby uczynić teksty przystępniejszymi dla tych użytkowników, którzy nie potrafią szybko czytać tekstu cerkiewnosłowiańskiego lub greckiego. Wszystkie wspomniane części projektu zostały szczegółowo opisane w rozdziale 2. Dodatek B zawiera dodatkowy opis formatów plików użytych w projekcie.

Nota licencyjna

Rozdział 1

Algorytmy dopasowania tekstów

Aby zbiór tekstów w różnych językach można było nazwać korpusem równoległym, teksty muszą być do siebie dopasowane. Dopasowanie[1] 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 [1]. 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:

[1] czasem też nazywane wyrównaniem, z ang. *alignment*

Input: $A = a_1, \ldots, a_n$
Input: $B = b_1, \ldots, b_m$
$cost \leftarrow$ macierz zerowa $n \times m$;
$cost_{0,0} \leftarrow 0$;
foreach $i \in 1 \,..\, n{-}1$ **do**
  foreach $j \in 1 \,..\, m{-}1$ **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` a_{i-1} `do` b_{j-1}
    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. Pominięte 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 $-\log p_k$, gdzie $p_k = D(a_i, b_j)$ jest prawdopodobieństwem, że zdanie a_i jest tłumaczeniem b_j, ozn. $a_i \sim b_j$. Zsumowany koszt dopasowania całego tekstu jest 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 $D(a_i, b_j)$.

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 (a_i, b_j) z przyporządkowanymi prawdopodobieństwami $\mathbb{P}(a_i \sim b_j | a_i, b_j)$, gdzie $a_i \sim b_j$ oznacza „a_i jest tłumaczeniem b_j".

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 nieznalezienia 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

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

Taka ocena odpowiada spostrzeżeniom z pracy [1], 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[2], 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:

$$\mathbb{P}(a_i \sim b_j | a_i, b_j) = \frac{\mathbb{P}(a_i \sim b_j, a_i, b_j)}{\mathbb{P}(a_i, b_j)} = \frac{\mathbb{P}(a_i \sim b_j)}{\mathbb{P}(a_i) + \ \mathbb{P}(b_j)} = \frac{\#((a_i, b_j))}{\#(a_i) + \#(b_j)},$$

gdzie # oznacza liczbę wystąpień w korpusie odpowiednio zdania lub pary zdań, a zapis $a_i \sim b_j$ oznacza zdarzenie „zdanie a_i jest tłumaczeniem zdania b_j”.

Tak obliczone prawdopodobieństwo jest bardzo sensownym wnioskiem, jednak okazuje się ono niepraktyczne, gdyż pamięć tłumaczeniowa zapytana o parę, w której będzie znała 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. wygładzanie[3]. W niniejszej pracy zastosowane zostało wygładzanie Laplace'a – wzór przyjmuje wtedy postać:

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

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. 1.1) – aby więc temu zaradzić, wszystkie akapity są przed przetwarzaniem zamienione na ciągi kluczy Metaphone[4].

1.4. Poprawianie dopasowań

Aby poprawić błędy w automatycznie wygenerowanym dopasowaniu, została opracowana następująca metoda: najpierw człowiek wskazuje zbiór par zdań, które powinny się znaleźć w dopasowaniu, a następnie algorytm dopasowuje fragmenty pomiędzy nimi. Poza – oczywiście – lepszym dopasowaniem, skraca to też w znaczący sposób czas działania programu dopasowującego, z powodu złożoności jego algorytmu $\mathcal{O}(n^2)$.

[2] Więcej o tym narzędziu można przeczytać w punkcie 2.2 i w internecie na stronie [12].

[3] ang. *smoothing*

[4] Dokładny opis tego algorytmu można znaleźć w punkcie 2.5.

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

1.5. 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) – analogicznie jak przy ręcznym poprawianiu dopasowań przedstawionym powyżej. Często to wstępne dopasowanie prowadzi również do poprawienia poprawności dopasowania. Bardzo dobitnie ukazuje to zjawisko rysunek 1.2.

1.6. Scalanie dopasowań

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

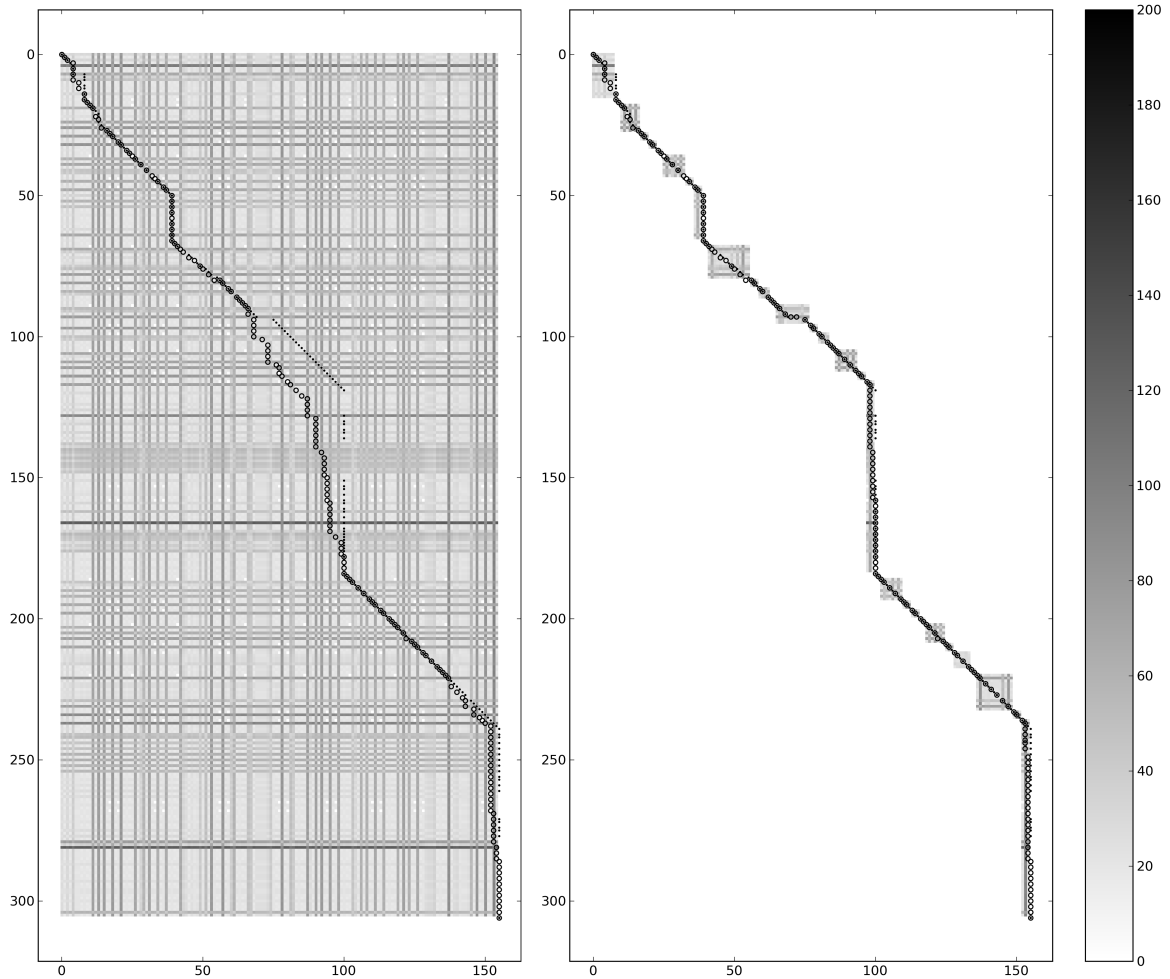
Prostym podejściem do tego zagadnienia jest wybranie z wyrównań tych par zdań, gdzie wszystkie dopasowania są zgodne. Jeśli dopasowania par tekstów są poprawne, w wyniku otrzymamy również poprawne dopasowanie.

Można to zapisać jako algorytm w następujący sposób:

Input: A_1 – dopasowanie tekstów 1 i 2 w formie par indeksów (patrz rys. 1.3)
Input: A_2 – dopasowanie tekstów 2 i 3 w formie par indeksów
Input: A_3 – dopasowanie tekstów 1 i 3 w formie par indeksów
$M_2 \leftarrow$ mapa utworzona z A_2, interpretowanego jako pary klucz-wartość ;
$A_r \leftarrow \emptyset$;
foreach $(i, j) \in A_1$ **do**
  if $(i, M_2[j]) \in A_3$ **then**
    $A_r \leftarrow A_r \cup \{(i, j, M_2[j])\}$;
  end
end
return A_r

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

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.

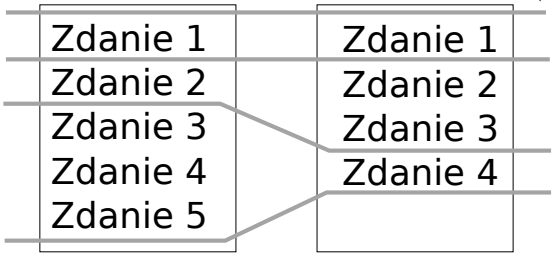
Ścieżka złożona z kółek biegnąca przez wykres oznacza wygenerowane dopasowanie tekstów – każdy jej punkt to przyporządkowanie sobie zdań. Ścieżka złożona z punktów oznacza poprawne, wzorcowe dopasowanie. Ścieżki nie są ciągłe w miejscach, gdzie mamy do czynienia z dopasowaniem wielu zdań (notabene, w 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).

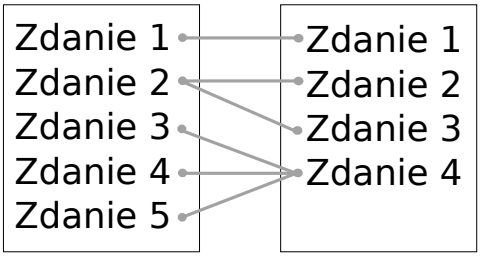
Wyniki dla lewego wykresu: precision: 67,30%, recall: 66,73%

Wyniki dla prawego wykresu: precision: 81,71%, recall: 89,22%

Rysunek 1.3: Pary indeksów dla przedstawionego dopasowania: $(0,0)$, $(1,1)$, $(2,3)$, $(5,4)$



Rysunek 1.4: Pary zbiorów indeksów dla przedstawionego dopasowania: $(\{1\},\{1\})$, $(\{2\},\{2,3\})$, $(\{3,4,5\},\{4\})$



1.7. Ewaluacja dopasowań

Ewaluacja dopasowania staje się łatwa, jeśli zamienimy dopasowanie na zbiór – wówczas można ocenić dopasowanie używając miar dokładności i pełności[5], szeroko stosowanych w systemach uczenia maszynowego. Aby jednak równomiernie rozłożyć „wagę" dopasowania na wszystkie fragmenty, potrzebne będą też wagi przy elementach (czyli parach indeksów) tego zbioru.

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

Input: A – dopasowanie tekstów w formie par zbiorów indeksów (patrz rys. 1.4)
$S \leftarrow \emptyset$;
$W \leftarrow \text{map} : \mathbb{N} \times \mathbb{N} \mapsto [0,1]$;
foreach (I,J) – para zbiorów indeksów z dopasowania A **do**
  if $I = \emptyset$ **then** $I \leftarrow \{\text{null}\}$;
  if $J = \emptyset$ **then** $J \leftarrow \{\text{null}\}$;
  $S \leftarrow S \cup (I \times J)$;
  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_a, S_b i wagi W_a, W_b dla odpowiednio wygenerowanego i wzorcowego dopasowania, możemy obliczyć dokładność i pełność dopasowania:

$$\text{precision} = \frac{\sum_{(i,j) \in S_a \cap S_b} W_a[i,j]}{|S_a|}, \quad \text{recall} = \frac{\sum_{(i,j) \in S_a \cap S_b} W_b[i,j]}{|S_b|}$$

[5]ang. *precision* i *recall*

Rozdział 2

Opis projektu

2.1. Konwersja danych wejściowych

Teksty pobrałem ze stron [2], [3] i [4]. Pliki z tych stron były odpowiednio w formatach HIP[1], 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 wiersze nie dłuższe niż 70 znaków, rozdzielając je pustymi wierszami (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 [12] 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, 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.

GIZA++ To narzędzie [11] służy do tworzenia dopasowania na poziomie słów z już dopasowanych par zdań. Nie użyłem go w mojej pracy, ponieważ przede wszystkim potrzebne

[1] HIP – standard zapisu tekstu cerkiewnosłowiańskiego. Więcej szczegółów w dodatku B.1

było dopasowanie na poziomie zdań. Dopasowanie na poziomie słów byłoby niewątpliwie ciekawym i użytecznym rozszerzeniem tej pracy, jest jednak poza jej zakresem.

Poliqarp Jest to wolny (i darmowy) zestaw narzędzi do przeszukiwania dużych korpusów [13]. 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 Poliqarpem, jednak teksty, którymi się zajmowałem w swojej pracy, nie były otagowane, więc nie mógłbym też wykorzystać pełni możliwości Poliqarpa. Poliqarp 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 skomplikowanym wzorcom odmiany języka cerkiewnosłowiańskiego [14] 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 [15], 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 katalogów

Cała implementacja została napisana w języku Python.

Główna funkcjonalność została umieszczona w katalogu **toolkit**. Wiele skryptów z tego folderu można wywołać z wiersza poleceń, aby wykonać różne zadania (patrz rozdział 2.7).

Interfejs internetowy znajduje się w katalogu **www**. Został on szczegółowo opisany w rozdziale 2.8.

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”.

W katalogu **data** znajduje się pamięć tłumaczeniowa, oznaczana wcześniej też symbolem TM.

Katalog **external** zawiera zewnętrzne biblioteki i narzędzia, z których korzysta projekt. Są to Hunalign [12] i Whoosh [17].

[2] „cu” oznacza język cerkiewnosłowiański, zgodnie ze standardem ISO 639-1.

2.4. Transformacje tekstu

Aby poprawnie wyświetlić tekst w języku cerkiewnosłowiańskim, potrzebna była specjalna czcionka, która oprócz liter zawierała również zestaw ligatur potrzebnych do poprawnego ułożenia znaków diakrytycznych nad konkretnymi literami. Przygotowanie tekstu do wyświetlenia w takiej czcionce również wymagało napisania prostego skryptu.

Język cerkiewnosłowiański w zapisie często występujących słów korzysta ze skrótów. Polegają one zazwyczaj na opuszczeniu samogłosek (zazwyczaj tytło proste, ◌҃) lub środkowej części wyrazu (zazwyczaj tytła literowe, ◌᷉ ◌᷊ ◌᷑). Język cerkiewnosłowiański korzysta ponadto z własnego systemu zapisu liczb. Wobec tego, aby dokonać transkrypcji fonetycznej takiego tekstu, należało najpierw rozwinąć wszystkie skróty i zamienić liczby na cyfry arabskie. Napisałem w tym celu moduł `translit.expand_cu`, zawierający listę 130 różnych skrótów i obsługujący liczby do 10 tysięcy (większe liczby nie występują w tekstach). Aby konwersja mogła działać wydajnie, skompilowałem wszystkie wzorce skrótów do jednego wyrażenia regularnego.

Następnie zaimplementowałem transkrypcję fonetyczną na polski w module `cu2pl`. Największy problem sprawiło mi tutaj dostosowanie wyniku do polskiej pisowni (przede wszystkim chodziło tu o litery „i” i „j” w różnych kontekstach).

Zaimplementowałem również transkrypcję fonetyczną z języka starogreckiego na polski. Dokonuję tej transkrypcji jednak zgodnie z regułami wymowy współczesnej greki, dlatego, że w ten sposób są zwykle czytane. Konwersja zachodzi w dwóch etapach: najpierw upraszczam zapis, usuwając znaki przydechu (dasia, prosgegrammeni, ypogegrammeni) i zamieniając wszystkie rodzaje akcentu (tonos, oxia, varia, perispomeni) na jeden (tonos), i dopiero wtedy dokonuję zamiany greckich liter i dyftongów na polskie odpowiedniki fonetyczne.

Wszystkie procedury obsługujące transformacje tekstu, w tym również te wspomniane powyżej, zostały zawarte w pakiecie `translit`.

2.5. Metaphone

Do rozmytego wyszukiwania według wymowy słów potrzebowałem algorytmu, działającego podobnie do Soundex [9], generującego dla słów klucze odpowiadające przybliżonej wymowie. Z jednej strony Soundex był do tego celu zbyt mało dokładny, z drugiej strony transkrypcja fonetyczna wszystkich tekstów na polski nie pozwoliłaby na znalezienie słowa, jeśli zostało błędnie wpisane w wyszukiwarkę.

Zainspirował mnie algotytm Metaphone [10]. Nie zastosowałem jednak dokładnie tego algorytmu, lecz stworzyłem jego wersję dostosowaną do moich danych[3]. Działanie algorytmu dla danego słowa przedstawia się następująco:

[3]W dalszej części pracy nazywam tę wersję po prostu „metaphone”

Input: w – słowo
Input: L – język podanego słowa (opcjonalny)
$w \leftarrow$ lowercase(w) ;
if w jest liczbą **then**
  return `"#"` $+ w$;
end
if $L =$ null **then**
  $L \leftarrow$ detect_language(w) ; `// wykrywanie języka`
end
if $L =$ `"cu"` **then**
  $w \leftarrow$ expand_cu(w) ; `// rozwijanie skrótów i zamiana liczb`
  if w jest liczbą **then**
    return `"#"` $+ w$
  end
end
foreach $(p, r) \in R_L$; `//` R_L `- tablica podstawień dla języka` L
do
  $w =$ replace(p, r, w) ; `// zamień` p `na` r `w napisie` w
end
$w' \leftarrow$ `""` ;
foreach $c \in w$; `//` c `- znak napisu` w
do
  if c jest nieznanym znakiem **then**
    return `"?"`
  end
  if c jest pierwszym znakiem $\vee\, c \notin \{$`"a"`, `"e"`, `"i"`, `"o"`, `"u"`$\}$ **then**
    $w' \leftarrow w' + c$;
  end
end
return w'
Przykładowe klucze Metaphone słów zostały przedstawione w tabeli 2.2.

2.6. Przeszukiwarka

Przeszukiwarka korpusu została zaimplementowana przy pomocy biblioteki Whoosh [17]. Aby możliwe było szybkie przeszukiwanie, najpierw tworzy się indeks bazujący na kluczach metaphone kolejnych słów tekstu, a następnie, przy wyszukiwaniu, otrzymujemy jako wyjście nazwę pliku i numer zdania, w którym zostało znalezione wyszukiwane słowo. Jets to używane do wyświetlenia kontekstu w wynikach wyszukiwania.

2.7. Interfejs wiersza poleceń

Większość modułów pakietu `toolkit` udostępnia swoją funkcjonalność poprzez prosty interfejs wiersza poleceń. 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 odpowiednio sformatowanej postaci) na standardowe wyjście.

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

metaphone	pl	cu	el
a	a	а	α
e	e, ę	є, ѐ, ѣ	ε, αι
i	i, y	ѷ, и, ы, і	ι, η, υ
o	o, ą	ѡ, о	ο, ω
u	u, ó	ү, оү	ου
b	b	б	μπ
c	c, ć, cz	ц, ч, щ	τσ
d	d, dz, dż	д	δ
f	f	ф, ѳ	φ, θ
h	g, h, ch	г, х	γ, χ
j	j	й	
k	k	к	κ
l	l	ль	λ
7	ł	л	
m	m	м	μ
n	n, ń	н	ν
p	p	п	π
r	r	р	ρ
s	s	с	σ, ς
t	t	т	τ
v	w	в, ѵ	β, υ
z	z, ź	з, ѕ	ζ
2	ż, rz, sz, ś	ж, ш	

Tabela 2.2: Przykładowe klucze Metaphone i przykłady słów im przyporządkowanych

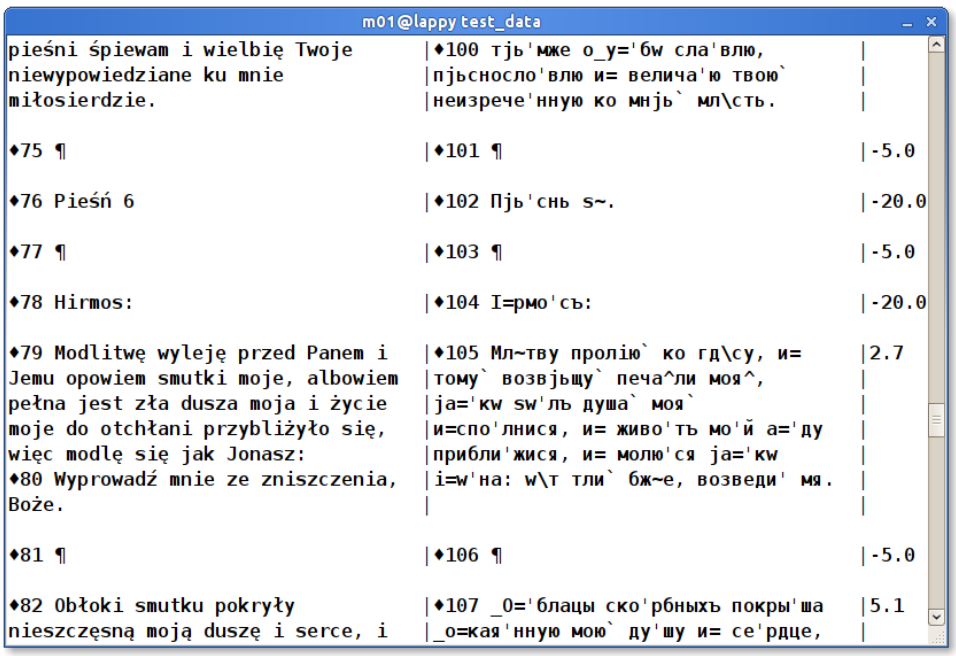
2c2lv	szczęśliwe szczęśliwy szczęśliwa
2l	szli żalu żyli ślę żal żale
7k	łk łuku łuk лꙋ̑ки лꙋ̑къ łąkę
avvkm	а҆ввакꙋ̑мъ ἀββακοὺμ
b2j	бж҃їѧ бж҃їю бѣжа́й bożej
d7h	długi długo dziełach до́лга до́лги
dbrvln	dobrowolnej dobrowolną dobrowolnymi
imnnts	ὑμνοῦντας ὑμνοῦντάς ὑμνοῦντές
hrnj	górnej го́рнїй го́рнѣй горѣ́нїѧ
isclv2	и҆сцѣлѣ́вшемꙋ и҆сцѣли́вшаѧ
j	ją ю҆̀ j ѧ ѧ҆̀ й je ja
km	κώμῃ комꙋ̀ кі́нма κημῷ ко́нма κῦμα
lhsms	λογισμούς λογισμοὺς λογισμοίς
ltrh	лїтꙋргі́и λειτουργία liturgią
mh7	mogły mgłę мглы mgłą мїхаи́лꙋ
mtr	μητέρα μητρὶ мт҃ри μέτρον мт҃ре
o	оу҆̀ ᾠ ᾧ ᾧ οἷά οἵου ο ο ὁ ὀ ὃ ὂ
ohnnnj	ѻ҆́гненныѧ ѻ҆́гненнаѧ ѻ҆́гненнѣй
pcc	począć pieczęci pieczęcie pieczęć
phd	pochodzi походѝ pochodzą pogodę
svc	свѣщꙋ̀ свѣщѝ свѣщѐ siewcy
t	та̀ τό тꙋ̀ ты̀ τω τῆ τῇ τού τῂ ті́и
t2	też toś tuż toż tyrze trzy то́же
zvr	ѕвѣ́рїє zawrę ѕвѣ́рь zawory

`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 [17].

`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. 1.2.

`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 2.1.

Rysunek 2.1: Przykładowe dopasowanie tekstów wyświetlone w konsoli.



Pakiet `translit` Wszystkie skrypty z pakietu `translit` można uruchomić z wiersza poleceń, aby dokonać odpowiedniej transliteracji na wybranym tekście. Jako argument należy podać plik z tekstem, lub `-`, aby korzystać ze standardowego wejścia. Dostępnie 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 dodatek B.1),
- `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 [16]. 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. 2.2.

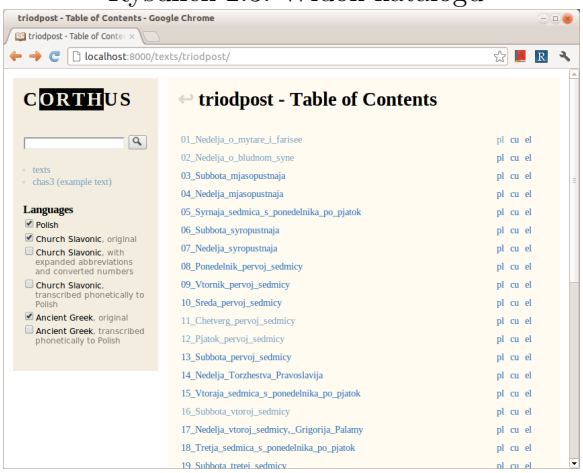
Rysunek 2.2: Widok tekstu w interfejsie internetowym

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

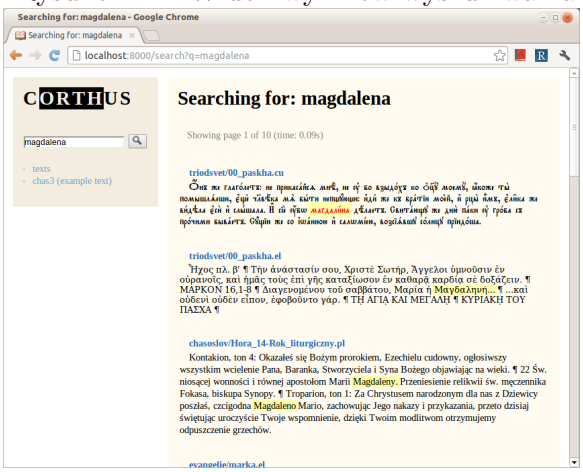
Widoki Cały serwis internetowy obsługują 3 widoki.

- **Widok tekstu** – Wyświetla tekst równoległy (widoczny na rys. 2.2). Po lewej stronie znajdują się pola wyboru języków, których zmiana powoduje przeładowanie strony z nowymi ustawieniami.
- **Widok katalogu** – Wyświetla spis treści książki lub spis książek. To zadanie sprowadza się do wyświetlenia odpowiednio sformatowanej listy plików (rys. 2.3).
- **Widok wyników wyszukiwania** – Wyświetla wyniki przeszukiwania korpusu (rys. 2.4).

Rysunek 2.3: Widok katalogu



Rysunek 2.4: Widok wyników wyszukiwania



Szablony Szablony są prostymi plikami HTML, korzystającymi z zewnętrznych arkuszy stylów CSS i skryptów w JavaScripcie. 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 (Firefox 3.6+, Opera 10.0+, Chrome 2.0+, Safari 4.0+, Internet Explorer 9+) obsługują ją poprawnie. Starsze

wersje Internet Explorera wyświetlają odstępy 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.

2.9. Zakończenie

Skromnym zdaniem autora, w pracy została zrealizowana większość celów, o których mowa we wprowadzeniu. Wiele detali wciąż wymaga dopracowania, jednak najważniejsza funkcjonalność została zaimplementowana i przetestowana. Jeśli stworzony serwis zostanie opublikowany w internecie, z pewnością pomoże to w jego dalszym rozwoju.

Dodatek A

Spis zawartości płyty

Na dołączonej płycie zostały zapisane następujące pliki i katalogi:

- `/praca_magisterska.pdf` – niniejsza praca,
- `/corthus/` – główny katalog projektu,
- `/corthus/toolkit/` – główna funkcjonalność projektu,
- `/corthus/www/` – interfejs internetowy,
- `/corthus/texts/` – baza tekstów,
- `/corthus/data/` – pamięć tłumaczeniowa,
- `/corthus/index/` – indeks wyszukiwania,
- `/corthus/external/` – zewnętrzne biblioteki i narzędzia.

Dodatek B

Formaty plików

B.1. HIP i UCS – tekst cerkiewnosłowiański

W czasach, kiedy więszość tekstów cerkiewnosłowiańskich została zdigitalizowanych, standard Unicode nie zawierał wszystkich znaków potrzebych do zapisu tekstu w języku cerkiewnosłowiańskim. W odpowiedzi na to powstał standard HIP (opisany dokładnie w internecie na stronie [5]). 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 B.1.

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 tytła literowe (znaki `2DE0-2DFF`) i literę ꙗ (`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” [6]. 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 tytłami 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* [7].

B.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 wedug standardu Unicode, ze względu na problemy z wyświetlaniem tych znaków (patrz punk B.1). 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.

Połamałem 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 B.1: Kodowanie znaków cerkiewnosłowiańskich w standardzie HIP

A a	ⰰ ⰰ
Б б	ⰱ ⰱ
В в	ⰲ ⰲ
Г г	ⰳ ⰳ
Д д	ⰴ ⰴ
Е е	ⰵ ⰵ
_Е _е <Е> <е>	ⰵ ⰵ ⰵ ⰵ
Ж ж	ⰶ ⰶ
S s	ⰸ ⰸ
З з	ⰷ ⰷ
И и	ⰻ ⰻ
І і	ⰼ ⰼ
_і <і>	ⰹ ⰹ
Й й	ⰽ ⰽ
К к	ⰾ ⰾ
Л л	ⰿ ⰿ
М м	ⱀ ⱀ
Н н	ⱁ ⱁ
О о	ⱂ ⱂ
_О _о <О> <о>	ⱃ ⱃ ⱃ ⱃ
W w	ⱆ ⱆ
_W _w <W> <w>	ⱇ ⱇ ⱇ ⱇ
П п	ⱈ ⱈ
Р р	ⱋ ⱋ
С с	ⱌ ⱌ
Т т	ⱍ ⱍ
У у	ⱎ ⱎ
_У _у <У> <у>	ⱏ ⱏ ⱏ ⱏ
О_У О_у о_у О<У> О<у> о<у>	ⱐ ⱐ ⱐ ⱐ ⱐ ⱐ
Ф ф	ⱑ ⱑ
Х х	ⱒ ⱒ
W\т w\т	ⱓ ⱓ
Ц ц	ⱔ ⱔ
Ч ч	ⱕ ⱕ
Ш ш	ⱖ ⱖ
Щ щ	ⱗ ⱗ
Ъ ъ	ⱘ ⱘ
Ы ы	ⱙ ⱙ
Ь ь	ⱚ ⱚ
Jь Jь jь	ⱛ ⱛ ⱛ
Ю ю	ⱜ ⱜ
JA Ja ja	ⱝ ⱝ ⱝ
Я я	ⱞ ⱞ
_КС _Кс _кс <КС> <Кс> <кс>	ⱟ ⱟ ⱟ ⱟ ⱟ ⱟ
_ПС _Пс _пс <ПС> <Пс> <пс>	ⱡ ⱡ ⱡ ⱡ ⱡ ⱡ
F f	Ɫ Ɫ
V v	ⱴ ⱴ

B.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.

Dodatek C

Licencja Apache 2.0

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. **Definitions**.

   "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

   "Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding

communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. **Grant of Copyright License**. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. **Grant of Patent License**. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. **Redistribution**. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. **Submission of Contributions**. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. **Trademarks**. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. **Disclaimer of Warranty**. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. **Limitation of Liability**. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. **Accepting Warranty or Additional Liability**. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

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, *Krótka 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`