

ZACHODNIOPOMORSKI UNIWERSYTET TECHNOLOGICZNY W SZCZECINIE
WYDZIAŁ INFORMATYKI

mgr inż. Marcin Pietras

**Syntaktyczna i semantyczna analiza danych tekstowych
z wykorzystaniem modeli Markowa realizowanych sprzętowo**

Rozprawa doktorska

Promotor: dr hab. inż. Przemysław Klęsk, prof. nadzw.

Pragnę serdecznie podziękować promotorowi
dr hab. inż. Przemysławowi Klęskowi, prof. nadzw.

Niniejszą rozprawę dedykuję mojej żonie Kamili Pietras.

Szczecin 2018



SIN. 7089

D.012012019

Pragnę serdecznie podziękować promotorowi
dr hab. inż. Przemysławowi Kłęskowi, prof. nadzw.

Niniejszą rozprawę dedykuję mojej żonie Kamili Pietras.

Spis treści

Spis treści

1	Wprowadzenie	2
1.1	Cel i teza pracy	4
1.2	Struktura pracy	4
1.3	Analiza syntaktyczna i semantyczna	5
1.4	Sposoby przypisywania wyrazom kategorii	7
1.4.1	Metody oparte o projekcje oznaczeń	7
1.4.2	Metody oparte o uczenie maszynowe z nadzorem	8
2	Ukryte modele Markowa	10
2.1	Struktura modelu	10
2.1.1	n -gramowa struktura obserwacji	12
2.1.2	Warstwowa struktura ukrytych modeli Markowa	13
2.2	Rozszerzona arytmetyka w dziedzinie logarytmicznej	14
2.3	Zadania w modelach Markowa i przebieg obliczeń	15
2.3.1	Algorytm Forward-Backward	16
2.4	Uczenie modelu	17
2.4.1	Reestymacja Bauma-Welcha	18
2.5	Dekodowanie ścieżki stanów	19
2.5.1	Algorytm Viterbiego	19
2.6	Reestymacja parametrów	21
2.6.1	Szacowanie prawdopodobieństw poprzez częstości	21
2.6.2	Algorytmy wygładzania	21
2.7	Aplikacja HMM-Toolbox	23
2.7.1	Tworzenie i zarządzanie modelami	23
2.7.2	Uczenie modeli HMM na podstawie zbiorów danych	24
3	Pozyskiwanie informacji z danych w formacie HTML	27
3.1	Aktualne trendy w pozyskiwaniu informacji ze źródeł internetowych	31
3.1.1	Rozwój technik pozyskiwania informacji ze źródeł internetowych	31
3.1.2	Sposób pozyskiwania informacji ze źródeł internetowych	32
3.1.3	Reprezentacja danych uzyskanych z formatu HTML	32
3.1.4	Typowa struktura stron internetowych	33
3.1.5	Proponowana metoda opisu informacji ze źródeł częściowo ustrukturyzowanych	34
3.2	Dane w formacie HTML jako sekwencja obserwacji dla ukrytego modelu Markowa	35
3.2.1	Wstępny parser formatu HTML	35
3.2.2	Przestrzeń międzytekstowa w formacie HTML	36
3.2.3	Mierzalne cechy przestrzeni tekstowej	36
3.3	Przygotowanie modelu HMM	37
3.4	Uczenie nadzorowane – znakowanie treści	38

3.4.1	Etykietowanie danych i procedura nauczania	38
3.5	Wyniki	41
4	Syntaktyczna analiza treści	44
4.1	Bank drzew składniowych	44
4.1.1	Penn Treebank	45
4.1.2	Składnica frazowa	46
4.1.3	Metody konstruowania drzew składniowych	46
4.2	Bank drzew zależności	49
4.2.1	Metody konstrukcji drzew zależności	50
4.2.2	Podejście oparte o wykres zależności	50
4.2.3	Podejście oparte o model przejść	51
4.3	Składnica – typy zależnościowe w języku polskim	51
4.3.1	Typy argumentowe	52
4.3.2	Typy nieargumentowe	53
4.3.3	Typy nieargumentowe (motywowane morfologicznie)	54
4.3.4	Typy koordynujące	54
4.4	Przygotowanie modeli HMM	55
4.4.1	Obserwacja zorientowana na afiksy	55
4.4.2	Wybór modeli i danych uczących	57
4.5	Ocena modeli	57
4.5.1	Model rozpoznawania kategorii syntaktycznych dla języka angielskiego	57
4.5.2	Model rozpoznawania grup frazowych dla języka angielskiego	61
4.5.3	Model rozpoznawania funkcji wyrazów w zdaniu dla języka angielskiego	64
4.5.4	Model rozpoznawania kategorii syntaktycznych dla języka polskiego	68
4.5.5	Model rozpoznawania kategorii zależnościowych dla języka polskiego	73
4.5.6	Podsumowanie i porównanie wyników	77
5	Opis na różnych poziomach semantyki leksykalnej	80
5.1	Modele semantyki dla danych tekstowych	80
5.1.1	Semantyka formalna	80
5.1.2	Funkcjonalny model semantyki	81
5.1.3	Leksykalny model semantyki	81
5.2	Odkrywanie pola semantycznego	82
5.2.1	Pole semantyczne na wyższym poziomie leksykalnym	84
5.2.2	Mapowanie na Wordnet	84
5.3	Przykłady użycia	86
6	Bezpieczeństwo numeryczne modeli Markowa	89
6.1	Lemat „o długości sekwencji niebezpiecznej numerycznie“	89
6.2	Przykłady numeryczne dla lematu	92
7	Akceleracja algorytmów uczenia maszynowego	97
7.1	Synteza wysokiego poziomu	98
7.1.1	Podstawowe informacje o HLS	99
7.1.2	Pętle w Vivado HLS	103
7.1.3	Tablice i praca z pamięcią	104
7.1.4	Użycie języków wysokiego poziomu	105
7.1.5	System Procesora (PS) i Logika Programowalna (PL)	106
7.2	Dostęp do pamięci, pamięć wieloportowa	107
7.3	Sprzętowa realizacja algorytmu Bauma-Welcha D&C	109
7.3.1	Jednostka przetwarzania stanów	109
7.3.2	Blok Forward-Backward D&C	111

7.3.3	Blok Gamma-Xi D&C	113
7.3.4	Blok Numerator-Denominator D&C	114
7.4	Sprzętowa realizacja algorytmu Viterbi D&C	117
7.4.1	Blok Argmax D&C	117
7.4.2	Blok nawracania D&C	119
7.5	MLEAU – jednostka aproksymacji wielu funkcji	120
7.5.1	LAU – aproksymacja funkcji logarytmicznej	121
7.5.2	EAU – aproksymacja funkcji wykładniczej	123
7.5.3	Tablice przeglądowe z równoległym dostępem	125
7.6	Wyniki implementacji	128
7.7	Synteza i wykorzystanie zasobów FPGA	129
7.8	Ocena wydajności	130
7.8.1	MLEAU	130
7.8.2	Wpływ zredukowania precyzji zmiennoprzecinkowej modeli na długość ba- danej sekwencji	131
8	Podsumowanie	132
	Spis rysunków	135
	Spis tabel	137
	Bibliografia	149
A	Zawartość płyty DVD	150
A.1	HMM-Toolbox	150
A.1.1	Projekt Netbeans	150
A.1.2	Ukryte modele Markowa	150
A.1.3	Bazy danych uczących	150
A.2	Banki Drzew	150
A.2.1	Penn Treebank	150
A.2.2	OntoNotes Release 5.0	151
A.2.3	2009 CoNLL Shared Task Part 2	151
A.2.4	Składnica frazowa i zależnościowa	151
A.3	Sprzętowa realizacja w FPGA	151
A.3.1	Projekt Vivado	151
A.3.2	Projekt HLS	151
A.3.3	Projekt RTL	151

7.3.3	Blok Gamma-Xi D&C	113
7.3.4	Blok Numerator-Denominator D&C	114
7.4	Sprzętowa realizacja algorytmu Viterbi D&C	117
7.4.1	Blok Argmax D&C	117
7.4.2	Blok nawracania D&C	119
7.5	MLEAU – jednostka aproksymacji wielu funkcji	120
7.5.1	LAU – aproksymacja funkcji logarytmicznej	121
7.5.2	EAU – aproksymacja funkcji wykładniczej	123
7.5.3	Tablice przeglądowe z równoległym dostępem	125
7.6	Wyniki implementacji	128
7.7	Synteza i wykorzystanie zasobów FPGA	129
7.8	Ocena wydajności	130
7.8.1	MLEAU	130
7.8.2	Wpływ zredukowania precyzji zmiennoprzecinkowej modeli na długość ba- danej sekwencji	131
8	Podsumowanie	132
	Spis rysunków	135
	Spis tabel	137
	Bibliografia	149
A	Zawartość płyty DVD	150
A.1	HMM-Toolbox	150
A.1.1	Projekt Netbeans	150
A.1.2	Ukryte modele Markowa	150
A.1.3	Bazy danych uczących	150
A.2	Banki Drzew	150
A.2.1	Penn Treebank	150
A.2.2	OntoNotes Release 5.0	151
A.2.3	2009 CoNLL Shared Task Part 2	151
A.2.4	Składnica frazowa i zależnościowa	151
A.3	Sprzętowa realizacja w FPGA	151
A.3.1	Projekt Vivado	151
A.3.2	Projekt HLS	151
A.3.3	Projekt RTL	151

Finansowanie prac badawczych

Prezentowane badania nad syntaktyczną i semantyczną analizą danych tekstowych z wykorzystaniem modeli Markowa realizowanych sprzętowo wykonane zostały przy wsparciu z dotacji celowej numer: 517-05-023-4781/17, przyznanej przez Ministerstwo Nauki i Szkolnictwa Wyższego.

Rozdział 1

Wprowadzenie

Analiza syntaktyczna i semantyczna odgrywają ważną rolę w automatyzacji przetwarzania danych tekstowych. Przeważnie rozpoznawanie kategorii syntaktycznych dokonywane jest w pierwszej kolejności, a na podstawie otrzymanej struktury składniowej wykonywana jest analiza semantyczna. Podobnie jest również w tej rozprawie, gdzie na podstawie otrzymanej struktury składniowej dokonywana jest analiza pola semantycznego wybranego wyrazu (ang. *Word of Interest*) w badanym tekście. Na przełomie ostatnich dwóch dekad naukowcy zaproponowali wiele systemów dokonujących syntaktyczno-semantycznej analizy o różnorodnej specjalizacji tematycznej (Bos 2011, Cambria & White 2014). Wciąż poszukiwane są coraz nowsze metody wykorzystujące techniki uczenia maszynowego zarówno z nadzorem, jak i bez. Największy postęp w tej dziedzinie współgra z popularnością języka, dla którego dany system jest rozwijany – nie dziwi więc fakt, że dla języka angielskiego dostępne są najbogatsze bazy danych (tj. PropBank (Palmer i in. 2006), NomBank (Meyers Adam & Macleod 2008), Penn Treebank (Taylor i in. 2003)) odpowiednio oznakowanych tekstów (syntaktycznie, ale i semantycznie). Dla mniej popularnych języków tworzone są również podobne bazy danych (np. Składnica) głównie za sprawą rozwoju technik parsowania i projekcji oznaczeń z innego języka (Pado & Lapata 2009). Nadal jednak oznaczanie danych tekstowych częściami mowy i częściami zdania jest skomplikowanym zadaniem wymagającym intensywnych obliczeń. Choć nowoczesne techniki często pozwalają osiągnąć wystarczającą dokładność analizy, to wciąż jest wiele do zrobienia w kierunku optymalizacji stosowanych metod.

Badania z zakresu lingwistyki komputerowej koncentrują się nad rozwiązaniami pozwalającymi na skuteczne przeprowadzenie wysoce dokładnej analizy tekstów zapisanych w języku naturalnym. W dynamicznie rozwijającej się dziedzinie uczenia maszynowego poszukuje się innowacyjnego podejścia wykonującego obliczenia w najefektywniejszy sposób. Dobrym przykładem jest tu dziedzina nauczania sztucznych sieci neuronowych przechodząca swój renesans za sprawą głębokiego uczenia (ang. *deep learning*), które potwierdziło swoją skuteczność w wielu dziedzinach rozpoznawania wzorców (Lecun i in. 2015, Schmidhuber 2015). Jednakże nowe podejście – choć skuteczne – jest wyjątkowo wymagające obliczeniowo (obliczanie odpowiedzi wielowarstwowej sieci liczącej często tysiące neuronów w każdej z warstw, w których przetwarzanie informacji dokonywane jest na różnych poziomach abstrakcji¹). Dlatego też wiele ośrodków badawczych i przemysłowych prowadzi intensywne badania w celu stworzenia nowych wydajnych rozwiązań sprzętowych (Gupta i in. 2015a, Zhang i in. 2015, Ovtcharov i in. 2015). Wykorzystując dedykowane rozwiązania sprzętowe możliwe jest osiągnięcie wysokiego stopnia zrównoleglenia algorytmów uczenia maszynowego (Pietras 2014a). Szczególne znaczenie ma to dla rozwiązań stosujących chmury obliczeniowe (ang. *cloud computing*). Przykładowo firma Google budowę zaplecza swoich usług z zakresu uczenia maszynowego i sztucznej inteligencji oparła na własnym wyspecjalizowanym chipie TPU (ang. *Tensorflow Processing Unit*) (Yunfei 2017), podczas

¹ Często etapy ekstrakcji i selekcji cech osadza się w konwolucyjnych sieciach neuronowych (czyli opartych na operacji mnożenia splotowego znanego z dziedziny przetwarzania obrazów), gdzie każdy neuron stanowi pewien filtr cech, stąd wiele cech przekłada się na dużą liczbę neuronów.

gdy firma Microsoft zdecydowała się na zastosowanie układów FPGA (ang. *Field-Programmable Gate Array* – bezpośrednio programowalna macierz bramek). Dzięki rekonfigurowalności² tych układów – czyli możliwości zmiany struktury obliczeniowej w trakcie pracy układu – łatwo przełączyć kontekst aplikacji. Oznacza to, że w jednej chwili możliwe jest na przykład przetwarzanie tekstu, a następnie rozpoznawanie obrazów przy wykorzystaniu fizycznie tej samej części zasobów FPGA.

W niniejszej pracy duża uwaga przywiązywana jest do efektywnej *sprzętowej realizacji* algorytmów programowania dynamicznego wykorzystywanych w metodach analizy danych tekstowych. Należy wyjaśnić, że pod pojęciem „realizacji sprzętowej” algorytmu rozumiane jest wykonanie algorytmu przy użyciu układu logicznego (zbiór bramek logicznych) lub obwodu elektronicznego, nie zaś tradycyjnego programu komputerowego uruchomionego na procesorze ogólnego przeznaczenia (CPU). W przypadku tej rozprawy wybrane algorytmy zrealizowane są przez odpowiednio połączone bramki logiczne w blokach układu cyfrowego FPGA z wykorzystaniem języka opisu sprzętu VHDL (ang. *Very High Speed Integrated Circuit Hardware Description Language*). Chcąc dokonać sprzętowej realizacji pewnych algorytmów trzeba zmierzyć się nie tylko z zagadnieniami zrównoleglania pętli czy ograniczeniami zasobów, ale przede wszystkim z problemami wynikającymi z utraty stabilności numerycznej (Pietras 2015, Gupta i in. 2015b, Hulden 2012). Obliczenia stosowane w różnych podejściach probabilistycznych (np. w modelach Markowa, sieciach bayesowskich) mają tendencję do utraty stabilności głównie ze względu na rodzaj wykonywanych obliczeń oraz własności maszynowych reprezentacji liczb (w szczególności reprezentacji wg standardu IEEE 754).

Prezentowane w rozprawie metody, wykorzystujące ukryte modele Markowa (ang. *Hidden Markov Models*), pozwalają na pozyskiwanie informacji ze źródeł internetowych w taki sposób, że dla podanego na wejściu adresu URL (ang. *Uniform Resource Locator*) strony internetowej analizowany jest jej kod HTML, w wyniku czego **rozpoznawana jest najpierw treść właściwa** zawartego na stronie artykułu (czyli odfiltrowane zostają elementy graficzne strony, reklamy, komentarze, forum dyskusyjne, podpisy rysunków, itp.), następnie **przeprowadzana jest syntaktyczna analiza** tej treści (oznaczanie częściami mowy i częściami zdania), i wreszcie dla wybranych istotnych informacji **tworzony jest opis na różnych poziomach semantyki leksykalnej**.

²Rozróżnia się rekonfigurację całościową i częściową układu FPGA oraz rekonfigurację statyczną (kiedy układ jest nieaktywny) i dynamiczną (podczas pracy układu) (Aithal 2016).

1.1 Cel i teza pracy

Głównym celem rozprawy jest zaprojektowanie ukrytych modeli Markowa, które pozwolą na analizę danych tekstowych poprzez algorytmy programowania dynamicznego realizowane sprzętowo.

W powyższym celu głównym mieszczą się następujące **cele szczegółowe**:

1. opracowanie metody pozyskiwania istotnych danych tekstowych ze źródeł HTML,
2. opracowanie metody oznaczania tekstu częściami mowy i częściami zdania,
3. opracowanie metody opisu istotnych informacji w tekście na różnych poziomach semantyki leksykalnej,
4. sprzętowa realizacja modeli Markowa z możliwością redukcji precyzji zmiennoprzecinkowej przy zachowaniu założonej dokładności.

Odnosnie drugiego punktu należy zaznaczyć, że oznaczanie tekstu częściami mowy (np. rzeczownik, czasownik, przymiotnik) i częściami zdania (np. podmiot, orzeczenie) nazywane jest bardziej formalnie odpowiednio anotacją morfosyntaktyczną oraz anotacją typami relacji zależnościowych.

Teza pracy jest sformułowana następująco:

Sprzętowe realizacje ukrytych modeli Markowa o zredukowanej reprezentacji zmiennoprzecinkowej pozwalają na analizę sekwencji tekstowych z zadowalającą dokładnością wykorzystując logarytmowane wersje algorytmów Viterbiego oraz Forward-Backward.

1.2 Struktura pracy

Rozprawa składa się z ośmiu rozdziałów, bibliografii, oraz załącznika. Pełny zakres pracy składa się z 151 stron, 58 ilustracji oraz 36 tabel.

W rozdziale pierwszym przedstawione zostały wyzwania z zakresu lingwistyki komputerowej oraz przykłady zastosowania dedykowanych rozwiązań sprzętowych w celu akceleracji obliczeń związanych z algorytmami uczenia maszynowego. W ramach wprowadzenia do analizy syntaktycznej i semantycznej przedstawiono przegląd opracowanych w tym celu systemów. Ponadto pokrótce opisano sposoby przypisywania kategorii syntaktycznych i semantycznych wyrazom w analizowanym tekście.

Rozdział drugi zawiera przegląd algorytmów wykorzystywanych w obliczeniach ukrytych modeli Markowa. Przedstawiona została w nim ogólna struktura zarówno modelu, jak i wykorzystywanych obserwacji. Ze względu na rozważania dotyczące stabilności numerycznej stanowiące istotny element niniejszej pracy, algorytmy związane z modelami Markowa zostały podane w tym rozdziale w dwóch wersjach: tradycyjnej (operującej na prawdopodobieństwach) i logarytmowanej (operującej na logarytmach prawdopodobieństw). Zaprezentowana również została metodologia nauczania i reestymacji³ parametrów modeli, ponadto pokrótce przedstawiono aplikację HMM-Toolbox, utworzoną na potrzeby badań i eksperymentów przeprowadzonych w ramach niniejszej pracy.

Trzeci rozdział dotyczy danych tekstowych pochodzących ze źródeł HTML. Opisane zostały w nim struktura języka HTML oraz sposoby reprezentacji danych. Przedstawiono historię i przegląd technologii pozyskiwania informacji. Zaprezentowana została autorska metoda (niezależna od języka) służąca do pozyskiwania istotnych danych tekstowych oraz rozpoznawania treści właściwej z wykorzystaniem ukrytych modeli Markowa. Obszernie opisana została także

³Pod pojęciem „reestymacji” rozumiane jest wielokrotne doszacowywanie parametrów modelu HMM na podstawie danych uczących zgodnie z procedurą Bauma–Welcha znaną pod nazwą „Baum–Welch reestimation” (Leonard E. Baum & Weiss 1970).

procedura przygotowania i nauczania modelu. Uzyskane wyniki kategoryzacji danych ze źródeł HTML podsumowują tę część pracy.

Czwarty rozdział poświęcony został analizie syntaktycznej i semantycznej. Przedstawiono w nim przegląd metod konstruowania drzew składniowych. Wykorzystując techniki nadzorowanego nauczania maszynowego, dokonane zostało rozpoznawanie kategorii syntaktycznych oraz typów relacji zależnościowych w tekstach w języku polskim, jak i w języku angielskim. Do wykonania tego zadania wykorzystano zbiory drzew składniowych, których opis zawarto w tym rozdziale (wraz z opisem zestawu rozpoznawanych kategorii syntaktycznych). Dodatkowo, z dużą szczegółowością przedstawione zostały także: n -gramowy model obserwacji zorientowanej na afiksy, sposoby konstruowania drzewa zależności oraz sposoby określania podstawowych funkcji w strukturze zdania. Z kolei na potrzeby analizy semantycznej utworzone zostały odrębne zestawy stanów modelu Markowa, opracowane na podstawie oznaczeń drzew zależnościowych „Składnica“ dla języka polskiego (Hajnicz 2014) i „PropBank“ dla języka angielskiego (Palmer i in. 2006). Wyczerpująco opisano procedurę nauczania warstwowych ukrytych modeli Markowa. W podsumowaniu rozdziału, wyniki otrzymane dla prezentowanych ukrytych modeli Markowa porównane zostały z wynikami raportowanymi w pracach naukowych: (Surdeanu i in. 2008), (Iwakura & Okamoto 2008) i (Wróblewska 2014, 2018)).

Rozdział piąty skupia się na opisie istotnych informacji, zawartych w tekście, na różnych poziomach semantyki leksykalnej. Pokróćce przedstawione zostały w nim główne cechy sieci semantycznej i poziomów semantyki leksykalnej. Zaprezentowano metody disambiguacji i dopasowywania wyrazów. Korzystając z zasobów Słownosieci (ang. *Wordnet*)⁴ oraz ustrukturyzowanej formy informacji powstałej w wyniku analizy syntaktycznej i semantycznej opracowana została metoda wielopoziomowego opisu relacji semantycznych występujących w tekście.

Szósty rozdział poświęcono określeniu ram stabilności numerycznej obliczeń. Zaprezentowano, a jednocześnie udowodniano możliwość występowania warunków zapewniających bezpieczeństwo numeryczne ukrytych modeli Markowa o zredukowanej reprezentacji liczbowej pozwalającej na analizę sekwencji tekstowych z zadowalającą dokładnością, wykorzystując logarytmowane wersje algorytmów programowania dynamicznego.

W rozdziale siódmym zaprezentowana została sprzętowa akceleracja algorytmów programowania dynamicznego wykorzystywanych w obliczeniach ukrytych modeli Markowa. Korzystając z technik syntezy wysokiego poziomu utworzono w układzie FPGA architekturę obliczeniową o wysokim stopniu zrównoleglenia opartą o jednostki przetwarzania stanów. Przy sprzętowej realizacji algorytmów wykorzystano metodologię projektową „dziel i zwyciężaj“. Dla wydajnego dekodowania ukrytych ścieżek w ramach modeli Markowa przedstawiono modyfikację algorytmu Viterbiego. Szczególna uwaga poświęcona została jednostce aproksymacji wielu funkcji logarytmicznych i wykładniczych „MLEAU“, która pozwala na wydajne obliczenia w dziedzinie logarytmicznej.

Pracę kończy podsumowanie zawierające wnioski na temat dokładności oraz skuteczności opracowanych metod.

1.3 Analiza syntaktyczna i semantyczna

Wielu badaczy (Lluis & Marquez 2008, Lluis 2008, Dai i in. 2009, Samuelsson i in. 2008, Llus i in. 2013) uznało, że system realizujący interakcje między analizą składniową i semantyczną umożliwia uzyskanie jakościowo lepszego wyniku niż w analogicznych systemach, w których te czynności są wykonywane oddzielnie.

Dzisiaj jest oczywiste, że z powodu niejasności w konstrukcji języka naturalnego (Przepiórkowski i in. 2012), budowa drzew składniowych z dużą dokładnością nie może ograniczać się tylko do znajomości gramatyki języka dotyczącej części mowy i cech morfologicznych. Niejednoznaczność języka naturalnego wymaga opracowania specjalnych analizatorów wykorzystujących metody

⁴Wordnet oraz Słownosieć to fizycznie różne (choć odpowiadające sobie) zasoby. Warto zaznaczyć, że twórcy powstałych później zasobów polskich – Słownosieci – również posługują się angielskim tłumaczeniem „Wordnet“. W związku z powyższym w niniejszej pracy te terminy używane są zamiennie.

statystyczne i podejścia z użyciem technik uczenia maszynowego. W szczególności, Przepiórkowski i in. (2012, str. 52) stwierdzają, że

„(...) W wypadku korpusów znacznie większych, takich jak należący obecnie do największych na świecie Narodowy Korpus Języka Polskiego, jedyną możliwością jest anotacja automatyczna, za pomocą specjalnie do tego stworzonych programów komputerowych. Programy takie muszą się jednak nauczyć znakować teksty, a do tego potrzebują. . . tekstów znakowanych przez człowieka. Stąd właśnie potrzeba stworzenia ręcznie anotowanego podkorpusu NKJP (...)”

oraz

„(...) Bezpośrednie sformalizowanie lingwistycznych reguł znakowania okazuje się zadaniem znacznie trudniejszym niż stworzenie takiego korpusu treningowego (...)”

Dlatego też w dzisiejszych parserach tekstów w języku naturalnym często analizowana jest nie tylko informacja o części mowy, cechach morfologicznych i cechach strukturalnych, ale także znaczenie leksykalne wyrazów, co w pewnym sensie jest formą znajomości semantyki języka (Baroni & Zamparelli 2010).

W obszarze przetwarzania tekstu języka naturalnego określenie semantycznej struktury jest jednym z głównych wyzwań w analizie sekwencji tekstowych. Zagadnienie znane jest również jako powierzchniowa analiza semantyczna (ang. *shallow semantic parsing*), do rozwoju której przyczyniły się prace D. Jurafsky'ego i D. Gildea (Gildea & Jurafsky 2002), którzy zastosowali zasoby PropBank (dla języka angielskiego). Model zasobów PropBank zakłada, że każda wartość słowa bazowego posiada zestaw określonych funkcji semantycznych. Do dalszego rozwoju metod analizy syntaktycznej i semantycznej przyczynili się uczestnicy konkursu CoNLL Shared Task 2008 (Surdeanu i in. 2008) oraz CoNLL Shared Task 2009 (Hajic i in. 2009). W 2008 roku analiza dotyczyła tylko języka angielskiego. W 2009 roku uczestnicy dokonali analizy sześciu kolejnych języków: katalońskiego, chińskiego, czeskiego, japońskiego, niemieckiego i hiszpańskiego. Sześć drużyn biorących udział w konkursie w 2008 roku i cztery zespoły w 2009 roku zastosowały metody, które w pewien sposób łączyły metody analizy składniowej i semantycznej.

W pracach (Lluis & Marquez 2008, Lluis 2008) opisano system, który został zgłoszony do konkursu w 2008 roku, a w pracy (Lluis i in. 2009) przedstawiono modyfikacje systemu, który brał udział w konkursie rok później. Syntaktyczna i semantyczna analiza w tych systemach była realizowana poprzez budowanie drzewa zależności równocześnie z utworzeniem etykiet relacji semantycznych. Do wstępnego i głównego parsera zastosowano algorytm Eisnera (Eisner 1996). Badania eksperymentalne nad tą metodą wykazały, że połączona analiza nie wpływa na wyniki parsera syntaktycznego, ale poprawia jakość rozpoznawania semantycznego w stosunku do systemu, w którym analizy syntaktyczna i semantyczna są wykonywane osobno.

System opisany w pracy (Henderson i in. 2008) przedstawia analizator korzystający z metody „shift-reduce”⁵ z równoległą budową konstrukcji składniowej i semantycznej zdania. W celu predykcji struktury zależności dla analizatora przeszkolony został model oparty o dynamiczną sieć Bayesa – zmieniającą swoją strukturę na podstawie wstępnie określonego stanu wyjściowego (Titov & Henderson 2007). Doświadczenia przeprowadzone w badaniach wykazały, że w przypadku braku współpracy pomiędzy analizą syntaktyczną i semantyczną znacznemu pogarszaniu ulega jakość analizy semantycznej. W pracach (Gesmundo i in. 2009, Henderson i in. 2013) autorzy wykazali również, że integracja analizy syntaktycznej i semantycznej może nieznacznie obniżać jakość tej pierwszej. W pracy (Samuelsson i in. 2008) zastosowano połączenie ośmiu parserów skonstruowanych na podstawie różnych konfiguracji systemu MaltParsera, co pozwoliło na znaczne zwiększenie jakości analizy w porównaniu z wynikami najlepszego pojedynczego

⁵Parser przesuwający (ang. *shift*) symbole na stos. Jeśli symbole ze stosu pasują do prawidłowej reguły gramatycznej w bieżącym kontekście, wówczas analizator składni redukuje (ang. *reduce*) je zastępując symbolem nieterminalnym.

analizatora. W pracy (Dai i in. 2009) wdrożone zostało podejście, w którym rozpoznawanie kategorii syntaktycznych i semantycznych wykonywane jest sekwencyjnie. Eksperymenty wykazały, że takie podejście może również znacznie poprawić efekt analizy. Autorzy sformułowali wniosek, że efekt dalszej analizy zależy od jakości wstępnego rozpoznawania kategorii syntaktycznych. Z kolei Sun i in. (2008) realizowali podejście, które łączy w sobie etapy wstępnej analizy składniowej i semantyczne poszukiwania argumentów za pomocą modelu Markowa z wykorzystaniem maksymalnej entropii (McCallum i in. 2000).

Wśród ostatnich publikacji dotyczących analizy składniowej i semantycznej, oprócz prac ukazujących rezultaty seminarium CoNLL Shared Task należy zauważyć pozycję (Llus i in. 2013), która proponuje podejście do maksymalizacji wspólnej oceny struktur składniowych i semantycznych. Doprowadziło to do bardziej precyzyjnego określenia funkcji semantycznych w porównaniu z układem, w którym te dwa typy analizy prowadzone są oddzielnie.

Jak widać, stosowane przez badaczy metody wykraczają daleko poza proste techniki zliczania tokenów, kolokacji i semantycznych klas słów na etapie analizowania. Niektórzy badacze proponują podejście alternatywne do tradycyjnego, kiedy to analiza syntaktyczna budująca drzewo składniowe ściśle poprzedza analizę semantyczną (która stanowi reprezentację znaczenia tekstu) i łączy te dwa rodzaje analiz w ramy jednej procedury. Istnieją parsery od razu budujące głębokie semantyczne struktury, bez struktur składniowych pośrednich – co całkowicie usuwa granice między analizą składniową i semantyczną (Grefenstette i in. 2014).

W rozprawie jako teoretyczną podstawę do reprezentowania semantyki tekstu zaproponowano autorską warstwową strukturę ukrytych modeli Markowa, gdzie na kolejnych poziomach przetwarzania odbywa się: pozyskiwanie treści właściwej, a następnie anotacja tej treści przypisująca wyrazom kategorii syntaktyczne i dalej kategorie zależnościowe. Tym samym analiza na wyższym poziomie posiłkuje się informacjami uzyskanymi z analizy na niższym poziomie przetwarzania. Model ten znajduje zastosowanie w wielu przypadkach przetwarzania tekstów języka naturalnego, w tym opisu semantycznego. Jako autorskie elementy opracowanego rozwiązania należy wskazać przede wszystkim: zaprojektowanie modeli HMM (czyli odpowiedni dla realizowanych zadań dobór przestrzeni stanów i obserwacji), a także propozycję technik prowadzenia obliczeń w ramach tych modeli z zachowaniem stabilności numerycznej (w szczególności przy ograniczonej precyzji zmiennoprzecinkowej). Dzięki powyższym możliwa jest dalsza – semantyczna – analiza tekstu, która nie zawiera już elementów nowatorskich i jest przeprowadzana z wykorzystaniem zasobów Słownosieci (Maziarz i in. 2016). Na tym etapie odbywa się identyfikacja funkcji semantycznej (Fillmore 1968), a w szczególności badanie pola semantycznego (Gao & Xu 2013) wybranych wyrazów. Gromadzony jest opis (synonimy, asocjacje, cechy i działania podmiotu, działania wobec podmiotu) z uwzględnieniem różnych relacji semantyki leksykalnej takich jak np.: hiperonimia, hiponimia i kohiponimia.

1.4 Sposoby przypisywania wyrazom kategorii syntaktycznych i semantycznych

Niniejszy punkt opisuje dwa główne podejścia do problemu określenia struktury syntaktycznej i semantycznej: (1) wykorzystujące projekcję oznaczeń, (2) wykorzystujące techniki uczenia maszynowego na odpowiednio oznakowanym zbiorze danych.

1.4.1 Metody oparte o projekcję oznaczeń

Wadą metod określania struktury z wykorzystaniem uczenia maszynowego z nadzorem jest wysoki poziom uzależnienia od dostępności i jakości oznaczonych korpusów (czyli oznakowanych baz danych językowych). W celu zapewnienia odpowiedniej jakości nauczania korpus musi być bardzo starannie przygotowany. Oznakowanie takich zasobów jest bardzo czasochłonne i kosztowne, dlatego w ostatnich latach naukowcy badają podejścia oparte na nienadzorowanym uczeniu maszynowym lub nadzorowanym tylko częściowo w celu zmniejszenia konieczności ręcznego oznaczania korpusów.

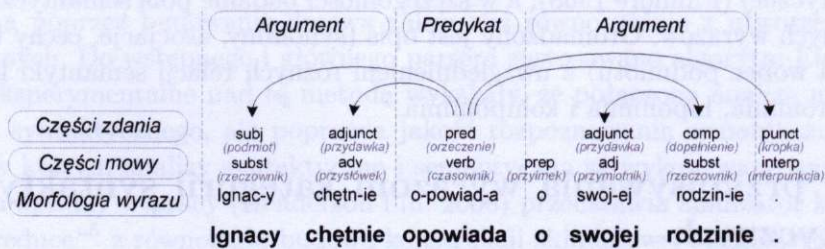
Szereg rozwiązań z częściowym udziałem nadzorowanego uczenia wykorzystuje dane uzyskane w wyniku projekcji oznaczeń. W tych podejściach semantyczna struktura oznaczonych zdań z jednego języka jest rzutowana na odpowiadającą frazę przetłumaczoną na inny język. Projekcje oznaczeń zostały przeprowadzone z powodzeniem wychodząc od korpusu języka angielskiego (oznaczonego semantycznie) i „rzutując go” na języki: niemiecki (Pado & Lapata 2009), szwedzki (Johansson & Nugues 2006), chiński (Fung & Chen 2004), francuski (Kaljahi i in. 2014) i polski (Wróblewska & Przepiórkowski 2014). Do określania struktur semantycznych na podstawie uczenia maszynowego stosuje się posilkowo metody klasteryzacji (Lang & Lapata 2014, 2011). Dokonywany jest podział wszystkich argumentów na grupy, do których przypisana jest odpowiednia funkcja semantyczna. Zestawy kategorii semantycznych charakterystycznych właściwości są generowane automatycznie w procesie grupowania.

1.4.2 Metody oparte o uczenie maszynowe z pełnym nadzorem

Metody określania struktury semantycznej oparte na nauczaniu w pełni nadzorowanym zostały rozwinięte w wielu pracach badawczych (Pradhan i in. 2004, Young-Sook i in. 2004, Toutanova i in. 2005, Choi & Palmer 2011). Przeprowadzono liczne eksperymenty dla różnych schematów rozkładu problemu na podzadania oraz badano różne podejścia algorytmiczne. Najbardziej powszechnym rozwiązaniem określania funkcji semantycznych jest podział zadania na kilka etapów:

- wyszukiwanie słów predykatów w zdaniu,
- wyszukiwanie słów argumentów w zdaniu,
- przypisanie funkcji do każdego z argumentów niezależnie od pozostałych,
- ocena i wybór funkcji semantycznej dla wszystkich argumentów predykatu.

Rysunek 1.1, wzorowany na diagramach z prac (Wróblewska & Przepiórkowski 2014, str. 27) i (Przepiórkowski i in. 2012, str. 124), przedstawia poglądowo składniową strukturę przykładowego zdania „Ignacy chętnie opowiada o swojej rodzinie” z uwzględnieniem zależności i morfologii wyrazów. W typowym prostym zdaniu predykat jest zwykle czasownikiem, natomiast argument reprezentuje fraza odrzeczownikowa.



Rysunek 1.1: Składniowa struktura zdania z uwzględnieniem morfologii wyrazów. (Źródło: opracowanie własne).

Wyszukiwanie predykatywnych słów w zdaniu jest zazwyczaj realizowane z wykorzystaniem słowników lub poprzez badanie cech morfologicznych (np. słowa predykatywne stanowią czasowniki). Na tym etapie jako klasyfikator do określenia części mowy, może być wykorzystywany zbiór reguł do ujednoznacznienia homonimów predykatów (np. niejednoznaczny wyraz „biegli” w zdaniu: *Oskarżeni biegli do sądu, aby biegli sądowi wydali orzeczenie.*) (Johansson & Nugues 2008, Bjorkelund i in. 2010). Etap określania semantycznego argumentu może wykorzystywać selekcję na podstawie zbioru prostych heurystyk. Przykładowo heurystyka zaproponowana w pracy (Thompson i in. 2003) może znacznie ograniczyć liczbę kandydatów w poszukiwaniu argumentów semantycznych w tekstach angielskich, co upraszcza zadanie dla klasyfikatora.

Początkowo zadanie identyfikacji argumentu jest traktowane jako problem grupowania elementów w kontekście drzewa składniowego – grupę kilku wyrazów trzeba prawidłowo przyporządkować jako argument. Dla każdego wybranego wyrazu i dla danego słowa bazowego klasyfikator przewiduje obecność relacji argument–predykat. W językach z dobrze rozwiniętą morfologią – takich jak słowiańskie – dodatkowo należy uwzględnić fleksyjne dopasowanie wyrazów, jak również funkcje związane z określeniem szyku wyrazów. Wyznaczanie argumentów i przypisywanie im semantycznej funkcji np. agent, obiekt, narzędzie (Fillmore 1968) może być łączone w jednym modelu uczenia maszynowego. Na etapie całościowej oceny dozwolonej semantycznej struktury sprawdzana jest zgodność z ograniczeniami modelu. Każdy argument może posiadać tylko jedną semantyczną funkcję i jednocześnie funkcja ta nie może być przypisana do innych argumentów będących w relacji z danym predykatem. Poszukiwana jest zatem taka struktura, która jest zgodna z określonym kryterium.

Warto zauważyć, że wiele metod określania funkcji semantycznej korzysta z analizy syntaktycznej. Wykazano, że jakość wyników analizy syntaktycznej ma duży wpływ na prawidłowe określenie funkcji semantycznej (Punyakanok i in. 2008, Ge & Mooney 2005). W pracy (Meza-Ruiz & Riedel 2009) wykorzystano model, oparty na logice Markowa, do identyfikacji słów bazowych, argumentów i predykatów. Istnieją również modele statystyczne używane do wspólnej oceny całej semantycznej struktury zdania. Model opisany w pracy (Cohn & Blunsom 2005), oparty na warunkowych polach losowych, dokonuje identyfikacji argumentów i przypisania im funkcji semantycznych. W pracy (Abney & Light 1999) do określenia predykatów i ich argumentów wykorzystano ukryty model Markowa.

Metody uczenia maszynowego z pełnym nadzorem potwierdziły w wielu różnych opracowaniach swoją skuteczność w wykonywaniu zadań związanych z rozpoznawaniem funkcji semantycznej. Prowadzenie prac dotyczących zastosowania metod uczenia maszynowego w celu określania semantyki dla tekstów w języku polskim jest utrudnione przez brak odpowiednio dużych oznaczonych korpusów, które mogłyby stanowić materiał uczący. Stąd i badań poświęconych problemowi automatycznego określania struktury semantycznej w tekstach w języku polskim jest niewiele. Do rozwoju tej dziedziny przyczynili się twórcy zasobów Składnicy Treebank (Hajnicz 2014) jak i Słownosieci (Maziarz i in. 2016), które to zasoby wykorzystane zostały również w tej rozprawie do maszynowego nauczania modeli i analizy semantycznej tekstów w języku polskim.

Rozdział 2

Ukryte modele Markowa

Rozważając sprzętową akcelerację algorytmów programowania dynamicznego wykorzystywanych w obliczeniach ukrytych modeli Markowa wprowadza się bardziej restrykcyjne ograniczenia co do reprezentacji liczbowej modelu. Jak już stwierdził Rabiner (1989), wykonywanie obliczeń typowych dla HMM szybko wyczerpuje precyzję (nawet podwójną) reprezentacji liczbowej. Jest to związane z koniecznością wykonywania długich sekwencji mnożenia wartości prawdopodobieństw (w pobliżu zera) dla przejść stanów i emisji obserwacji. W efekcie może dojść do sytuacji, gdzie pewien interesujący nas iloczyn prawdopodobieństw (matematycznie niezerowy) zeruje się numerycznie i w konsekwencji nie pozwala na prawidłowe porównania z innymi iloczynami. Można co prawda dokonać znaczącej poprawy stabilności przez wprowadzenie współczynników skalujących dla metody Forward-Backward jak to pokazano w tutorialu (Rabiner 1989), jednakże zabieg ten określić można formą łagodzenia objawów, a nie przyczyn niestabilności obliczeniowej. Istnieje jeszcze inne podejście – polegające na logarytmowaniu wielkości indukcyjnych występujących w algorytmach programowania dynamicznego dla modeli Markowa. Z uwagi na przeciwdziałanie funkcji logarytm oraz na fakt, że iloczyn zamieniamy na sumę, wyczerpywanie precyzji zmiennoprzecinkowej przebiega znacznie wolniej i w praktyce (przy podwójnej precyzji) eliminuje groźbę utraty stabilności numerycznej. Rabiner (1989) zaproponował realizację algorytmu Viterbiego w dziedzinie logarytmicznej. Z kolei logarytmowane wersje pozostałych algorytmów (Forward-Backward oraz reestymacja Bauma-Welcha) zostały wyczerpująco omówione w pracy (Mann 2006). Stabilna numerycznie realizacja tych algorytmów możliwa jest również w wariancie sprzętowym, dlatego też w niniejszej rozprawie od samego początku wszystkie algorytmy związane z HMM przedstawione zostaną również w wersji rozszerzonej, opracowanej dla dziedziny logarytmów. Stosowana notacja oznaczeń we wzorach matematycznych jest zgodna z tą przedstawioną w tutorialu Rabinera.

2.1 Struktura modelu

W ogólnym przypadku model Markowa przedstawić można jako graf skierowany, którego wierzchołki (w liczbie N) reprezentują zbiór stanów modelu $S = \{S_1, S_2, \dots, S_N\}$, a krawędzie są skojarzone z prawdopodobieństwami a_{ij} przejść pomiędzy stanami (dla $1 \leq i, j \leq N$). Tym samym wszystkie prawdopodobieństwa przejść można przechowywać w macierzy $N \times N$ w postaci tradycyjnej lub zlogarytmowanej:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{pmatrix}, \quad \log A = \begin{pmatrix} \log a_{11} & \log a_{12} & \dots & \log a_{1N} \\ \log a_{21} & \log a_{22} & \dots & \log a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \log a_{N1} & \log a_{N2} & \dots & \log a_{NN} \end{pmatrix}. \quad (2.1)$$

W rozważaniach podstawa logarytmu jest pomijana, ponieważ wszystkie logarytmy są proporcjonalne do siebie z dokładnością do stałego czynnika, natomiast na poziomie implementacji

można się posługiwać np. logarytmem o podstawie 2. W *ukrytym* modelu Markowa poza stanami wyróżniamy także obserwacje¹, które są emitowane (generowane) przez stany i tym samym obserwowalne na zewnątrz. W tym czasie same stany pozostają nieznane – ukryte. Niech $V = \{V_1, \dots, V_M\}$ oznacza zbiór wszystkich możliwych obserwacji. Prawdopodobieństwo wystąpienia obserwacji V_k (dla $1 \leq k \leq M$) pod warunkiem, że system przebywa w stanie S_i będzie oznaczane jako $b_i(k)$ (zapis $b_i(k)$ jest utożsamiany z $b_i(V_k)$). Zestaw wszystkich takich prawdopodobieństw reprezentuje macierz B , którą (podobnie jak w przypadku macierzy A) na potrzeby implementacji można przechowywać w formie zlogarytmowanej:

$$B = \begin{pmatrix} b_1(1) & b_1(2) & \dots & b_1(M) \\ b_2(1) & b_2(2) & \dots & b_2(M) \\ \vdots & \vdots & \ddots & \vdots \\ b_N(1) & b_N(2) & \dots & b_N(M) \end{pmatrix}, \quad \log B = \begin{pmatrix} \log b_1(1) & \log b_1(2) & \dots & \log b_1(M) \\ \log b_2(1) & \log b_2(2) & \dots & \log b_2(M) \\ \vdots & \vdots & \ddots & \vdots \\ \log b_N(1) & \log b_N(2) & \dots & \log b_N(M) \end{pmatrix}. \quad (2.2)$$

Uzupełnieniem modelu Markowa stanowi informacja o rozkładzie prawdopodobieństwa dla chwili początkowej:

$$\pi = (\pi_1 \quad \pi_2 \quad \dots \quad \pi_N), \quad \log \pi = (\log \pi_1 \quad \log \pi_2 \quad \dots \quad \log \pi_N). \quad (2.3)$$

Zestaw tych parametrów w pełni opisuje ukryty model Markowa zarówno w postaci tradycyjnej $\lambda = (A, B, \pi)$, jak i logarytmicznej $\lambda_{\log} = (\log A, \log B, \log \pi)$.

Model HMM wykorzystywany jest w celu znalezienia wyjaśnienia dla sekwencji obserwacji $O = (o_1, o_2, \dots, o_T)$, gdzie $o_t \in V$, za pomocą przypisania ciągu stanów $Q = (q_1, q_2, \dots, q_T)$, gdzie $q_t \in S$.

W celu lepszego przybliżenia Czytelnikowi przedstawionych wielkości oraz przyjętej notacji rozważmy następujący szkolny przykład. Powiedzmy, że rozpatrujemy pewien model zajęć wakacyjnych wyimaginowanego Jasia Nowaka, w którym wyróżniamy trzy stany pogody ($N = 3$):

$$S = \begin{matrix} \text{słońce} & \text{chmury} & \text{deszcz} \\ S_1 & S_2 & S_3 \end{matrix}. \quad (2.4)$$

a „na tle” tychże stanów rozpatrujemy następujący zbiór pięciu możliwych zajęć – obserwacji ($M = 5$) – które Jaś wykonuje:

$$V = \begin{matrix} \text{wędkowanie} & \text{rower} & \text{bieganie} & \text{krzyżówki} & \text{szachy} \\ V_1 & V_2 & V_3 & V_4 & V_5 \end{matrix}. \quad (2.5)$$

Przypuśćmy dalej, że ukryty model Markowa dla zajęć wakacyjnych Jasia został zidentyfikowany i ma postać:

$$A = \begin{pmatrix} 0.6 & 0.3 & 0.1 \\ 0.1 & 0.5 & 0.4 \\ 0.3 & 0.3 & 0.4 \end{pmatrix}, \quad B = \begin{pmatrix} 0.2 & 0.5 & 0.2 & 0.0 & 0.1 \\ 0.1 & 0.3 & 0.0 & 0.2 & 0.4 \\ 0.1 & 0.1 & 0.0 & 0.3 & 0.5 \end{pmatrix}, \quad \pi = (0.6 \quad 0.2 \quad 0.2). \quad (2.6)$$

Model ten wykazuje naturalną tendencję polegającą na tym, że w dni słoneczne Jaś preferuje zajęcia na dworze, zaś w dni deszczowe zajęcia domowe. Z drugiej strony model w uproszczeniu odzwierciedla uwarunkowania pogodowe w miejscu zamieszkania Jasia, gdzie, jak można zauważyć, występuje dość często pogoda deszczowa. Niemniej, powyższe stwierdzenie to tylko pewne ogólne tendencje procesu losowego, a szczegóły tkwią w konkretnych rozkładach prawdopodobieństwa. Jak wspomniano w poprzednim akapicie taki model może być wykorzystany w celu wyjaśnienia pewnej sekwencji obserwacji. I tak, przykładowe zadanie obliczeniowe przy znanym modelu HMM może wyglądać następująco. Powiedzmy, że podano nam zaobserwowaną sekwencję zajęć Jasia z kolejnych siedmiu dni wakacji ($T = 7$):

$$O = \begin{matrix} \text{(krzyżówki,} & \text{wędkowanie,} & \text{rower,} & \text{rower,} & \text{szachy,} & \text{krzyżówki,} & \text{bieganie)} \\ o_1 & o_2 & o_3 & o_4 & o_5 & o_6 & o_7 \end{matrix} \quad (2.7)$$

¹W przypadku jawnego (a nie ukrytego) modelu Markowa obserwacje są utożsamiane ze stanami.

i postawiono pytanie: *jaka jest najbardziej prawdopodobna sekwencja stanów pogodowych tworząca powyższą sekwencję obserwacji?* W sensie matematycznym jest to pytanie o tzw. *ścieżkę Viterbiego*, a sposób wyznaczania jej z wykorzystaniem programowania dynamicznego jest opisany w podrozdziale 2.5.1. Dla powyższego zadania okazuje się, że ścieżka ta ma postać:

$$Q = \underset{q_1}{(\text{deszcz}, \text{ słońce}, \text{ słońce}, \text{ słońce}, \text{ chmury}, \text{ deszcz}, \text{ słońce})} \underset{q_7} . \quad (2.8)$$

a jej prawdopodobieństwo wynosi: 9.24%. Liczba ta może zdawać się małą wartością prawdopodobieństwa. Należy jednak uświadomić sobie, że jest to maksymalna wartość wśród prawdopodobieństw wszystkich wchodzących do konkurencji możliwych ścieżek o długości $T = 7$, które mogły wygenerować daną sekwencję obserwacji. Dla jasności notacyjnej warto także zwrócić uwagę, że S i V reprezentują *zbiory* odpowiednio stanów oraz obserwacji (i mogą być rozumiane jako słowniki unikalnych możliwości), podczas gdy Q i O oznaczają *sekwencje* (lub równoważnie ciągi) nad dyskretnym czasem reprezentujące konkretne realizacje kolejnych wyrazów generowanych zgodnie z określonymi przez HMM rozkładami prawdopodobieństw. Innymi słowy, dziedziną dla każdego z wyrazów w ciągu q_t jest zbiór S , zaś dziedziną dla każdego z wyrazów w ciągu o_t jest zbiór V .

2.1.1 n -gramowa struktura obserwacji

Powszechnie stosowane w bioinformatyce i lingwistyce obliczeniowej n -gramy określają pewien model stochastyczny spełniający właściwość Markowa, służący do prognozowania kolejnych elementów łańcucha. Dla zadanej obserwowanej sekwencji $O = (o_1, o_2, \dots, o_T)$ możliwe jest utworzenie n -gramowej struktury przez uwzględnienie n przeszłych obserwacji. Najprostsza 1-gramowa (unigramowa) struktura traktuje wszystkie elementy sekwencji rozłącznie:

$$O_{n=1} = (o_1, o_2, o_3, \dots, o_T) . \quad (2.9)$$

Natomiast każda n -gramowa struktura dla $n \geq 2$ dokonuje łączenia obserwacji w coraz to dłuższe sekwencje, co przedstawiają poniższe zapisy.

- Obserwacja 2-gramowa:

$$O_{n=2} = (o_1 o_2, o_2 o_3, o_3 o_4, \dots, o_{T-1} o_T) . \quad (2.10)$$

- Obserwacja 3-gramowa:

$$O_{n=3} = (o_1 o_2 o_3, o_2 o_3 o_4, o_3 o_4 o_5, \dots, o_{T-2} o_{T-1} o_T) . \quad (2.11)$$

- Obserwacja n -gramowa:

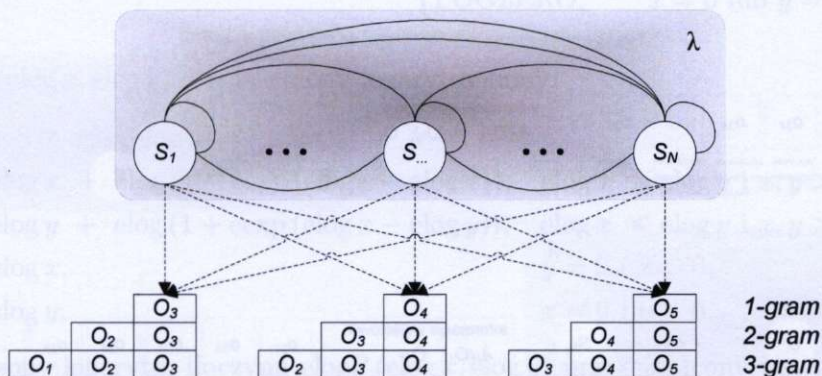
$$O_n = (o_1 o_2 \dots o_n, o_2 o_3 \dots o_{(n+1)}, o_3 o_4 \dots o_{(n+2)}, \dots, o_{(T-n+1)} \dots o_T) . \quad (2.12)$$

Wraz z zwiększaniem rzędu n -gramowej struktury rośnie liczba rozróżnialnych elementów pozyskanych z obserwowanej sekwencji. W wielu aplikacjach takie podejście potwierdziło pozytywny wpływ na skuteczność modelowania złożonych ciągów czasowych (Ziółko & Skurzok 2010). W ukrytym modelu Markowa warstwa obserwacji jest odseparowana od modelu, więc n -gramowa struktura obserwacji modyfikuje wyłącznie macierz emisji B . Dla przykładu, stosując w analizie tekstu obserwacje 1-, 2- i 3-gramową wyrażenia „piękne kwiaty” byłby reprezentowany poprzez następujące ciągi:

- obserwacja 1-gramowa (p, i, e, k, n, e, -, k, w, i, a, t, y),
- obserwacja 2-gramowa (pi, ie, ek, kn, ne, e-, _k, kw, wi, ia, at, ty, y-),

- obserwacja 3-gramowa (pię, ięk, ękn, kne, ne_, e.k, _kw, kwi, wia, iat, aty, ty_, y_).

Rysunek 2.1 prezentuje w sposób poglądowy ukryty model Markowa operujący n -gramową strukturą obserwacji.

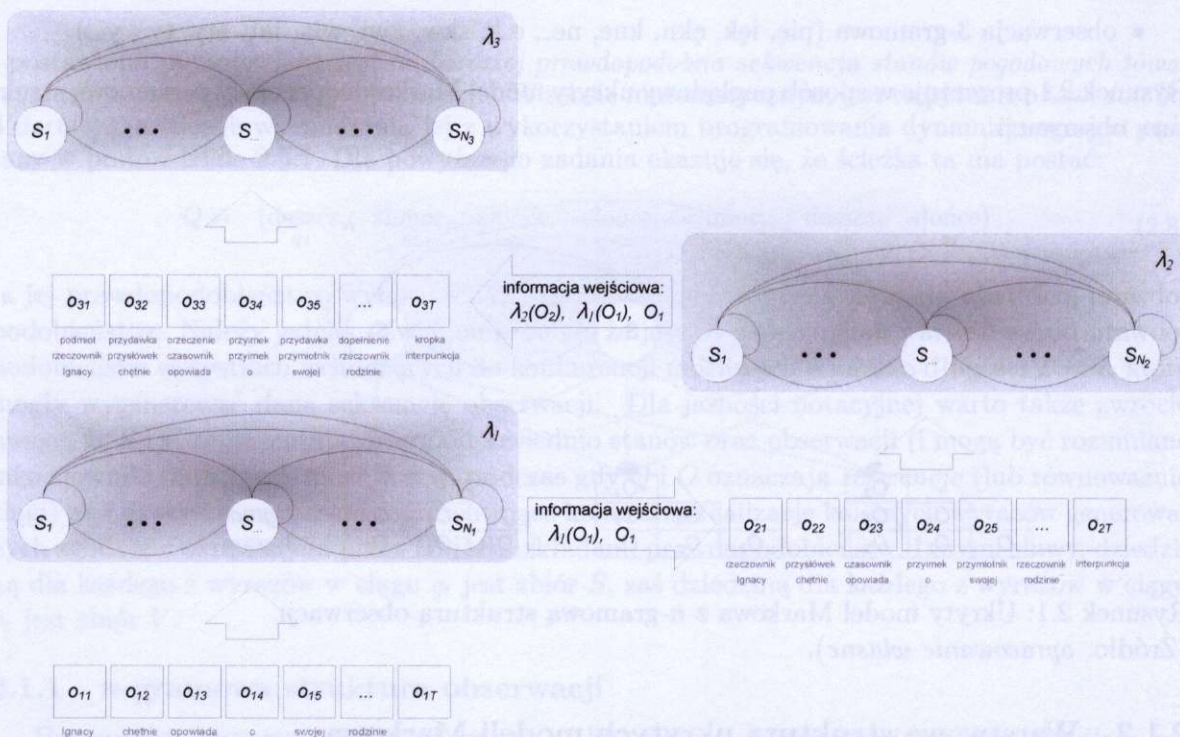


Rysunek 2.1: Ukryty model Markowa z n -gramową strukturą obserwacji.

(Źródło: opracowanie własne).

2.1.2 Warstwowa struktura ukrytych modeli Markowa

Ukrytość modeli Markowa może przybierać charakter warstwowy, szczególnie kiedy z każdym kolejnym poziomem reprezentowane są przez model coraz bardziej abstrakcyjne cechy. W literaturze przedmiotu spotkać można definicje i przykłady wykorzystania warstwowej (hierarchicznej) struktury ukrytych modeli Markowa. Nie ma jednak ograniczeń co do rodzaju obserwacji. Ogólnie założyć można, że warstwowa topologia stosowana jest, aby wyodrębnić pewne cechy, stąd w skład obserwacji modelu na poziomie $l + 1$ będzie wchodziła ścieżka stanów odkryta przez model na poziomie l . Oznacza to, że model na wyższym poziomie jest zależny od wyników modelu na niższym poziomie. Mimo tej relacji model na poziomie wyższym nadal może korzystać z tych samych obserwacji (lub w pewien sposób zredukowanych), co model na poziomie niższym. Rysunek 2.2 wizualizuje warstwową strukturę przetwarzania ukrytych modeli Markowa na potrzeby syntaktycznej i semantycznej analizy. Na rysunku wyróżnić można trzy różne modele Markowa (λ_1 , λ_2 i λ_3) operujące na trzech kolejnych poziomach przetwarzania. Ścieżka obserwacji dla każdego poziomu wyróżniona została przez dolny indeks (na pierwszym poziomie obserwacji O_1 , na drugim O_2 , na trzecim O_3). Obserwacja na wyższym poziomie wzbogacona jest informacją o stanach zdekodowanych przez konkretny model dla ścieżki obserwacji z poziomu niższego.



Rysunek 2.2: Warstwowa struktura przetwarzania ukrytych modeli Markowa.
 (Źródło: opracowanie własne).

W wyniku badania sekwencji obserwacji, model na pierwszym poziomie rozpoznaje dla każdego wyrazu kategorię syntaktyczną reprezentowaną w zdaniu. Informacja ta włączana jest do obserwacji na poziomie drugim, gdzie w wyniku dekodowania odkrywane są typy zależności zachodzące pomiędzy wyrazami w zdaniu. Na kolejnych poziomach możliwe jest zastosowanie modeli Markowa do rozpoznawania coraz to bardziej abstrakcyjnych znaczeń analizowanego tekstu.

2.2 Rozszerzona arytmetyka w dziedzinie logarytmicznej

Jak łatwo zauważyć parametry ukrytego modelu Markowa określone tradycyjnie jako prawdopodobieństwa mogą przybierać wartości równe zero. Stąd, tak jak to pokazał Mann (2006), do obliczeń należy wprowadzić specjalną obsługę operacji logarytmu z zera, jak i rozszerzoną arytmetykę liczb w dziedzinie logarytmicznej. W niniejszej pracy przyjęta została nomenklatura zgodna z tą wykorzystywaną przez Manna, dlatego też rozszerzone operacje logarytmiczne będą oznaczane prefiksem „e” (ang. *extended*), a specjalna obsługa logarytmu z zera oznaczona będzie mnemonikiem LOGZERO. W docelowej implementacji symbol LOGZERO można zastąpić pewną szczególną wartością przewidzianą przez dany typ liczbowy, np. ujemną nieskończonością lub symbolem „NaN” określającym wartość niereprezentowalną. Warto podkreślić, że w prezentowanych wzorach, aby poprawić czytelność, pomijane są nawiasy dla funkcji jednoargumentowych, np. $\text{elog } x$ zamiast $\text{elog}(x)$. W algorytmach programowania dynamicznego powiązanych z modelami Markowa można zdefiniować następujące cztery przydatne funkcje:

- $\text{eexp } x$ – rozszerzona funkcja wykładnicza:

$$\text{eexp } x = \begin{cases} e^x, & x \in \mathbb{R}; \\ 0, & x = \text{LOGZERO}. \end{cases} \quad (2.13)$$

- $\text{elog } x$ – rozszerzona funkcja logarytmująca:

$$\text{elog } x = \begin{cases} \log x, & x > 0; \\ \text{LOGZERO}, & x = 0. \end{cases} \quad (2.14)$$

- $\text{elogP}(\text{elog } x, \text{elog } y)$ – rozszerzony logarytm iloczynu działający na argumentach będących logarytmami:

$$\text{elogP}(\text{elog } x, \text{elog } y) = \begin{cases} \text{elog } x + \text{elog } y, & x \in \mathbb{R}, y \in \mathbb{R}; \\ \text{LOGZERO}, & x = 0 \text{ lub } y = 0. \end{cases} \quad (2.15)$$

- $\text{elogS}(\text{elog } x, \text{elog } y)$ – rozszerzony logarytm sumy:

$$\text{elogS}(\text{elog } x, \text{elog } y) = \begin{cases} \text{elog } x + \text{elog}(1 + \exp(\text{elog } y - \text{elog } x)), & \text{elog } x > \text{elog } y \text{ i } x, y > 0, \\ \text{elog } y + \text{elog}(1 + \exp(\text{elog } x - \text{elog } y)), & \text{elog } x \leq \text{elog } y \text{ i } x, y > 0, \\ \text{elog } x, & y = 0 \text{ i } x > 0, \\ \text{elog } y, & x = 0 \text{ i } y > 0. \end{cases} \quad (2.16)$$

Rozszerzony logarytm iloczynu $\text{elogP}(\text{elog } x, \text{elog } y)$ przekształcony jest na sumę rozszerzonych logarytmów, natomiast logarytm sumy $\text{elogS}(\text{elog } x, \text{elog } y)$ wymaga określenia, który z argumentów jest większy, aby ich różnica dała wartość ujemną. Funkcja wykładnicza dużych wartości ujemnych dąży do zera, a po przekroczeniu precyzji reprezentacji liczbowej występuje niedomiar i zwracanym wynikiem jest zero. Zabieg ten odgrywa ważną rolę w utrzymaniu stabilności numerycznej obliczeń.

O ile logarytm iloczynu (w szczególności iloczynu prawdopodobieństw) jest dość powszechnie spotykany i zamieniany na sumę logarytmów, tak logarytm sumy jest spotykany raczej rzadko. W kontekście modeli Markowa pojawi się on w sytuacji, gdy potrzebne będzie obliczenie sumy dwóch prawdopodobieństw, przy czym argumenty nie będą dane bezpośrednio a za pomocą logarytmów. Innymi słowy chcemy obliczyć $x + y$, gdzie $x = P(A)$ i $y = P(B)$ (prawdopodobieństwa pewnych zdarzeń), mając do dyspozycji $\log x$ oraz $\log y$. Co więcej, chcielibyśmy, aby zwróconym wynikiem był także logarytm tzn. $\log(x + y)$. Poniższe wyprowadzenie uzasadnia postać wzoru (2.16):

$$\begin{aligned} \log(x + y) &= \log\left(x \frac{(x + y)}{x}\right) \\ &= \log x + \log\left(1 + \frac{y}{x}\right) \\ &= \log x + \log(1 + \exp(\log y - \log x)). \end{aligned} \quad (2.17)$$

2.3 Zadania w modelach Markowa i przebieg obliczeń

Jednym z zadań stawianych w ramach modeli HMM jest obliczenie całkowitego prawdopodobieństwa danej sekwencji obserwacji $O = (o_1, \dots, o_T)$ dla znanego modelu, tj. znanej trójki $\lambda = (A, B, \pi)$. Stosując metodę bezpośrednią, gdzie wyznaczane jest prawdopodobieństwo dla wszystkich możliwych ścieżek stanów realizowanych przez zadany HMM dla badanej sekwencji obserwacji, można wyrazić to prawdopodobieństwo całkowite wzorem:

$$P(O|\lambda) = \sum_Q P(Q|\lambda)P(O|Q, \lambda), \quad (2.18)$$

gdzie $Q = (q_1, \dots, q_T)$ oznacza ścieżkę stanów ($q_t \in S$), a suma odbywa się po wszystkich możliwych ścieżkach. Dla ustalonych N i T liczba składników tej sumy wynosi N^T , a zatem rośnie wykładniczo ze względu na długość sekwencji. Przebieganie wszystkich ścieżek w wyczerpujący sposób – choć skuteczne – jest dość mało efektywne i posiada wykładniczą złożoność obliczeniową, dlatego też w praktycznych obliczeniach wykorzystywany jest bardziej wyrafinowany algorytm o nazwie Forward-Backward.

2.3.1 Algorytm Forward-Backward

Algorytm Forward-Backward ma złożoność $O(N^2T)$, czyli liniową ze względu na długość sekwencji T . Definiowane są dwie wielkości indukcyjne α i β , których sens probabilistyczny jest następujący:

$$\alpha_t(i) = P(o_1, \dots, o_t, q_t = S_i | \lambda), \quad (2.19)$$

$$\beta_t(i) = P(o_{t+1}, \dots, o_T | q_t = S_i, \lambda). \quad (2.20)$$

Wielkości te są następnie wyznaczane według następującego algorytmu.

1. Inicjalizacja

$$\alpha_1(i) = \pi_i b_i(o_1), \quad (2.21)$$

$$\beta_T(i) = 1. \quad (2.22)$$

2. Indukcja

dla $t = 2, 3, \dots, T$:

$$\alpha_t(j) = \sum_{i=1}^N \alpha_{t-1}(i) a_{ij} b_j(o_t), \quad (2.23)$$

dla $t = T - 1, T - 2, \dots, 1$:

$$\beta_t(i) = \sum_{j=1}^N a_{ij} b_j(o_{t+1}) \beta_{t+1}(j). \quad (2.24)$$

3. Zakończenie

$$P(O|\lambda) = \sum_{i=1}^N \alpha_T(i). \quad (2.25)$$

$$P(O|\lambda) = \sum_{i=1}^N \pi_i \beta_1(i). \quad (2.26)$$

W postaci tradycyjnej kolejne wartości α i β dla badanej sekwencji obserwacji otrzymywane są w wyniku mnożenia parametrów HMM z wartościami uzyskanymi w kroku poprzedzającym. Dla długich sekwencji działanie to może prowadzić do wyczerpania precyzji reprezentacji liczbowej, tym samym do utraty stabilności numerycznej obliczeń.

W postaci logarytmowanej użyty zostanie zapis elogS z indeksowaniem analogicznym do notacji Σ dla sumy. Zapis ten należy rozumieć jako sekwencyjne wykonanie funkcji elogS wg wzoru (2.16) – najpierw zsumowanie dwóch pierwszych składników, a następnie dodawanie składników trzeciego, czwartego, itd. do bieżącej sumy.

Logarytmowana wersja algorytmu Forward-Backward może być zapisana następująco:

1. Inicjalizacja

$$\text{elog } \alpha_1(i) = \text{elogP}(\text{elog } \pi_i, \text{elog } b_i(o_1)), \quad (2.27)$$

$$\text{elog } \alpha_t(i) = \text{LOGZERO}, 1 < t \leq T, \quad (2.28)$$

$$\text{elog } \beta_T(i) = 0, \quad (2.29)$$

$$\text{elog } \beta_t(i) = \text{LOGZERO}, 1 \leq t < T. \quad (2.30)$$

2. Indukcja

dla $t = 1, 2, \dots, T$:

$$\text{elog } \alpha_t(i) = \text{elogP}\left(\text{elog } b_i(o_t), \text{elogS}_{j=1}^N \text{elogP}(\text{elog } \alpha_{t-1}(j), \text{elog } a_{ij})\right), \quad (2.31)$$

dla $t = T - 1, T - 2, \dots, 1$:

$$\text{elog } \beta_t(i) = \text{elogS}_{j=1}^N \text{elogP}\left(\text{elog } a_{ij}, \text{elogP}(\text{elog } b_i(o_{t+1}), \text{elog } \beta_{t+1}(j))\right). \quad (2.32)$$

3. Zakończenie

$$P(O|\lambda) = \text{eexp}\left(\text{elogS}_{i=1}^N \text{elog } \alpha_T(i)\right). \quad (2.33)$$

Wyliczanie logarytmowanych wielkości α i β dokonywane jest z wykorzystaniem rozszerzonej arytmetyki logarytmów. Wraz z każdym kolejnym krokiem indukcji wartości kolejnych wielkości α i β otrzymywane są w wyniku akumulacji poprzednich obliczeń. Szczególnie pomocna jest tu funkcja elogS wykonująca logarytm sumy. Sumowanie w przeciwieństwie do wymnażania wyczerpuje dostępną bitową precyzję reprezentacji liczbowej znacznie wolniej, dzięki czemu wynik pozostaje znacznie dłużej stabilny numerycznie.

2.4 Uczenie modelu

Podczas wyliczania modelu określane jest prawdopodobieństwo, z jakim HMM o znanych parametrach $\lambda = (A, B, \pi)$ realizuje badaną sekwencję obserwacji $O = (o_1, o_2, \dots, o_T)$. W wielu zadaniach parametry HMM nie są jednak znane od razu. W niektórych przypadkach można je łatwo estymować (w sposób nadzorowany), jeżeli dysponuje się odpowiednio dużym zbiorem danych, w którym sekwencje obserwacji podane są wraz z towarzyszącymi stanami. Gdy taka komfortowa sytuacja nie ma miejsca, wówczas konieczne jest iteracyjne uczenie modelu HMM, które znane jest pod nazwą *reestymacji Bauma-Welcha*. Procedurę można rozpocząć od modelu losowego (spełniającego probabilistyczny warunek sumowania się wierszy macierzy do jedności). Następnie w ramach reestymacji, aktualny model zostaje zaktualizowany do nowego w taki sposób, aby maksymalizować prawdopodobieństwo wygenerowania rozważanej sekwencji.

2.4.1 Reestymacja Bauma-Welcha

Algorytm przez Bauma-Welcha korzysta z wyników algorytmu Forward-Backward i definiuje kolejne dwie wielkości indukcyjne. Pierwszą z nich jest $\gamma_t(i)$, która reprezentuje dla badanej sekwencji obserwacji prawdopodobieństwo $P(q_t = S_i | O, \lambda)$, czyli prawdopodobieństwo zdarzenia, że system w chwili t był w stanie S_i pod warunkiem zaistniałej obserwacji.

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^N \alpha_t(j)\beta_t(j)}. \quad (2.34)$$

Druga wielkość $\xi_t(i, j)$ dla badanej sekwencji obserwacji reprezentuje $P(q_t = S_i, q_{t+1} = S_j | O, \lambda)$ czyli prawdopodobieństwo zdarzenia, że z chwili t na chwilę $t + 1$ nastąpiło przejście ze stanu S_i do stanu S_j w świetle zaistniałej obserwacji.

$$\xi_t(i, j) = \frac{\alpha_t(j)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\sum_{x=1}^N \sum_{y=1}^N \alpha_t(x)a_{xy}b_y(o_{t+1})\beta_{t+1}(y)}. \quad (2.35)$$

Aktualizacja parametrów odbywa się na podstawie wartości oczekiwanych. Można przyjąć, że prawdopodobieństwo a_{ij} pozwala się oszacować poprzez stosunek oczekiwanej liczby przejść $S_i \rightarrow S_j$ do oczekiwanej liczby pobyków w S_i . W terminach wielkości γ i ξ można powyższy stosunek przedstawić jako

$$a_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)}, \quad (2.36)$$

z kolei prawdopodobieństwo emisji można oszacować jako

$$b_j(k) = \frac{\sum_{t=1}^T \gamma_t(j)}{\sum_{t=1}^T \gamma_t(j)}. \quad (2.37)$$

Dla dziedziny logarytmów wielkość γ wyliczana jest z wykorzystaniem rozszerzonej arytmetyki logarytmów, stąd określona będzie przedrostkiem „elog” jako $\text{elog } \gamma$:

$$\text{elog } \gamma_t(i) = \text{elogP}\left(\text{elogP}\left(\text{elog } \alpha_t(i), \text{elog } \beta_t(i)\right), -\text{elogS} \text{elogP}\left(\alpha_t(j), \text{elog } \beta_t(j)\right)\right). \quad (2.38)$$

Analogicznie, z wykorzystaniem rozszerzonej arytmetyki logarytmów, wyliczana jest wielkość ξ określona jako $\text{elog } \xi$:

$$\begin{aligned} \text{elog } \xi_t(i, j) = & \text{elogP}\left(\text{elogP}\left(\text{elogP}\left(\text{elog } \alpha_t(i), \text{elog } a_{ij}\right), \text{elog } b_j(o_{t+1})\right), \text{elog } \beta_{t+1}(j)\right), \\ & - \text{elogS} \text{elogS} \text{elogP}\left(\text{elogP}\left(\text{elogP}\left(\text{elog } \alpha_t(x), \text{elog } a_{xy}\right), \text{elog } b_y(o_{t+1})\right), \text{elog } \beta_{t+1}(y)\right). \end{aligned} \quad (2.39)$$

Wartość początkowa dla γ i ξ wynosi zero, zatem w dziedzinie logarytmów wartością początkową zarówno dla $\text{elog } \gamma_t(i)$, jak i dla $\text{elog } \xi_t(i, j)$ jest odpowiednik logarytmu z zera określony przez LOGZERO.

Następnym krokiem jest aktualizacja parametrów wg wzorów przedstawionych poniżej.

- Początkowe prawdopodobieństwo wystąpienia stanu:

$$\pi_i = \text{eexp} \log \gamma_1(i). \quad (2.40)$$

- Prawdopodobieństwo przejścia:

$$\text{numeratorA}(i, j) = \prod_{t=1}^{T-1} \text{elogS} \log \xi_t(i, j), \quad (2.41)$$

$$\text{denominatorA}(i) = \prod_{t=1}^{T-1} \text{elogS} \log \gamma_t(i), \quad (2.42)$$

$$a_{ij} = \text{eexp} \log P(\text{numeratorA}(i, j), -\text{denominatorA}(i)). \quad (2.43)$$

- Prawdopodobieństwo emisji:

$$\text{numeratorB}(j, k) = \prod_{t=1, o_t=V_k}^T \text{elogS} \log \gamma_t(j), \quad (2.44)$$

$$\text{denominatorB}(j) = \prod_{t=1}^T \text{elogS} \log \gamma_t(j), \quad (2.45)$$

$$b_j(k) = \text{eexp} \log P(\text{numeratorB}(j, k), -\text{denominatorB}(j)). \quad (2.46)$$

Na tym etapie warto podkreślić, że logarytmowane prawdopodobieństwa macierzy przejść i emisji wyliczane są jednorazowo przed przystąpieniem do kroku indukcji. Powrót z dziedziny logarytmów (jeżeli zajdzie taka potrzeba) dokonywany jest przez przekształcenie wykładnicze (2.43) i (2.46).

2.5 Dekodowanie ścieżki stanów

Dla badanej sekwencji obserwacji $O = (o_1, o_2, \dots, o_T)$ i HMM o znanych parametrach $\lambda = (A, B, \pi)$ należy określić optymalną sekwencję stanów $Q = (q_1, q_2, \dots, q_T)$. Wynik dekodowania ścieżki w dużym stopniu zależy od wybranego kryterium optymalności. Możliwe jest podejście zorientowane na pojedyncze stany, gdzie każdej obserwacji w badanej sekwencji przypisany jest ten stan, który generuje ją z największym prawdopodobieństwem statycznym w danej chwili t . Biorąc jednak pod uwagę relacje przejść między stanami, optymalne zdekodowanie ścieżki stanów sprowadza się do maksymalizacji prawdopodobieństwa kolejnych prefiksów (z algorytmu Forward), co może być realizowane za pomocą algorytmu Viterbiego.

2.5.1 Algorytm Viterbiego

Algorytm programowania dynamicznego zaproponowany przez Andrew Viterbiego jest szeroko stosowany w systemach telekomunikacji (Seshadri & Sundberg 1994), przetwarzaniu i syntezie mowy, bioinformatyce i lingwistyce komputerowej. Algorytm ten w swej implementacji podobny jest do algorytmu Forward, z tą różnicą że w przypadku Viterbiego szukana jest wartość maksymalizująca prawdopodobieństwo $P(Q|O, \lambda)$, że model przy wyemitowaniu o_t przebył ścieżkę stanów $q_1, \dots, q_t$. W tym celu definiowana jest zmienna indukcyjna δ , której sens probabilistyczny jest następujący:

$$\delta_t(i) = \max_{q_1, \dots, q_{t-1}} P(q_1, \dots, q_t = i, o_1, \dots, o_t | \lambda). \quad (2.47)$$

Dla odtworzenia (przy końcowym nawracaniu) przebytej ścieżki stanów stosowana jest pomocnicza tablica ψ , która przechowuje indeksy stanów otrzymanych w algorytmie drogą indukcji (indeksy zwracane przez operator $\arg \max$). Algorytm Viterbiego daje się wyrazić zarówno w postaci tradycyjnej lub zlogarytmowanej.

Algorytm Viterbiego w postaci tradycyjnej ma poniższą postać.

1. Inicjalizacja:

$$\delta_1(i) = \pi_i b_i(o_1). \quad (2.48)$$

2. Indukcja dla $2 \leq t \leq T$:

$$\delta_t(j) = \max_{1 \leq i \leq N} (\delta_{t-1}(i) a_{ij}) b_j(o_t), \quad (2.49)$$

$$\psi_t(j) = \arg \max_{1 \leq i \leq N} (\delta_{t-1}(i) a_{ij}). \quad (2.50)$$

3. Zakończenie:

$$P^* = \max_{1 \leq i \leq N} (\delta_T(i)), \quad (2.51)$$

$$q^*_T = \arg \max_{1 \leq i \leq N} (\delta_T(i)). \quad (2.52)$$

4. Nawracanie dla $t = T - 1, T - 2, \dots, 1$:

$$q_t^* = \psi_{t+1}(q_{t+1}^*). \quad (2.53)$$

Natomiast w postaci zlogarytmowanej algorytm Viterbiego ma poniższą postać.

1. Inicjalizacja:

$$\text{elog } \delta_1(i) = \text{elog } \pi_i + \text{elog } b_i(o_1). \quad (2.54)$$

2. Indukcja dla $2 \leq t \leq T$:

$$\text{elog } \delta_t(j) = \max_{1 \leq i \leq N} (\text{elog } \delta_{t-1}(i) + \text{elog } a_{ij}) + \text{elog } b_j(o_t), \quad (2.55)$$

$$\psi_t(j) = \arg \max_{1 \leq i \leq N} (\text{elog } \delta_{t-1}(i) + \text{elog } a_{ij}). \quad (2.56)$$

3. Zakończenie:

$$P^* = \exp \max_{1 \leq i \leq N} (\text{elog } \delta_T(i)), \quad (2.57)$$

$$q^*_T = \arg \max_{1 \leq i \leq N} (\text{elog } \delta_T(i)). \quad (2.58)$$

4. Nawracanie dla $t = T - 1, T - 2, \dots, 1$:

$$q_t^* = \psi_{t+1}(q_{t+1}^*). \quad (2.59)$$

Jak już zauważył to Rabiner, algorytm Viterbiego pracujący w dziedzinie logarytmicznej wymaga mniej obliczeń (ze względu na możliwość wcześniejszego jednorazowego wyliczenia logarytmowanych odpowiedników parametrów dla ukrytego modelu Markowa oraz liczenia sum zamiast iloczynów, co jest mniej kosztowną operacją), przy jednoczesnym zachowaniu wysokiej stabilności numerycznej.

2.6 Reestymacja parametrów

Wykorzystując nauczanie nadzorowane, możliwe jest empiryczne określenie parametrów ukrytego modelu Markowa, dla którego zadana sekwencja obserwacji została wygenerowana. W uczeniu nadzorowanym wymagana jest baza danych zawierająca oznakowanie – takie, że każdej obserwacji (w sekwencji obserwacji $O = (o_1, o_2, \dots, o_T)$) przyporządkowany jest określony stan (w sekwencji stanów $Q = (q_1, q_2, \dots, q_T)$), tworząc dwie równoliczne sekwencje. Warto podkreślić, że dobrze jest, kiedy znaczniki przypisane są dla kompletnej, w dodatku obszernej bazy danych. Nie jest to jednak warunek konieczny. Uczenie nadzorowane można przeprowadzić dla częściowo oznakowanej bazy tak jak to przedstawiono w pracy (Hovy & Hovy. 2012) otrzymując wstępnie wyuczony model, którego douczanie może być przeprowadzone na wiele sposobów np. dokonując treningu Viterbiego, lub stosując algorytm Bauma-Welcha.

2.6.1 Szacowanie prawdopodobieństw poprzez częstości

Częstość zdarzenia określić można na podstawie liczności jego wystąpień (zapisywanej dalej poprzez $\mathcal{L}$) względem liczności wszystkich zdarzeń w rozpatrywanej bazie danych. Na potrzeby rozważań symbol $\rightarrow$ oznaczać będzie przejście między stanami, natomiast symbol $\rightsquigarrow$ oznaczać będzie emisję obserwacji z danego stanu. Dokonując zliczania wystąpień każdej obserwacji względem wystąpień wybranego stanu, możliwe jest przybliżenie rozkładu prawdopodobieństwa emisji dla tego stanu.

$$b'_i(k) = \frac{\mathcal{L}(S_j \rightsquigarrow V_k)}{\sum_l \mathcal{L}(S_j \rightsquigarrow V_l)}. \quad (2.60)$$

Analogicznie, aby odtworzyć prawdopodobieństwa przejść między stanami w modelu Markowa, należy dokonać zliczania liczby przejść pomiędzy wybranymi stanami.

$$a'_{ij} = \frac{\mathcal{L}(S_i \rightarrow S_j)}{\sum_k \mathcal{L}(S_i \rightarrow S_k)}. \quad (2.61)$$

Powodzenie prawidłowego odtworzenia parametrów HMM w dużej mierze zależy od jakości danych uczących. W praktyce rzadko zdarza się, że baza danych w pełni pokrywa wszystkie możliwe kombinacje wystąpień stanów i obserwacji. Może się zdarzyć, że dana obserwacja czy też przejście między danymi stanami nie wystąpiło podczas nauczania modelu, model nie jest więc w stanie prawidłowo obsłużyć podobnego zdarzenia w przyszłości i już podczas testowania może objawiać niedostatki w uczeniu. Takiej sytuacji można zaradzić stosując odpowiedni algorytm wygładzania, jak to pokazano w pracy (Hazem & Morin 2013). Ponadto zachowując sumy normalizujące, możliwe jest douczenie modelu w przyszłości poprzez przeprowadzenie zliczania tylko dla nowej porcji danych.

2.6.2 Algorytmy wygładzania

Skuteczność estymacji parametrów przez zliczanie częstotliwości wystąpień zdarzeń względem określonych stanów silnie zależy od jakości danych uczących, a przede wszystkim od liczności wystąpień obserwacji. Szczególnym przypadkiem jest estymacja prawdopodobieństwa dla zdarzeń, które nie wystąpiły podczas nauczania. Prawdopodobieństwo wystąpienia takiej obserwacji wynosi więc zero, co przy wyliczaniu ścieżki stanów (gdzie występuje mnożenie prawdopodobieństw) utrudnia prawidłowe określenie prawdopodobieństwa na podstawie wyników doświadczenia (przez wyzerowanie wartości pojedynczego kroku). Równie uciążliwa jest estymacja prawdopodobieństwa dla zdarzeń, które występowały zawsze podczas nauczania i pozornie wydają się zdarzeniem pewnym. Stosowanie algorytmów wygładzania wprowadza odpowiednia poprawkę przy estymacji parametrów modelu, taką że niedostatek danych uczących w mniejszym stopniu wpływa na jakość uczenia modelu.

Poprawka Laplace'a

Jednym z najczęściej stosowanych algorytmów wygładzania jest algorytm Laplace'a zwany również „Add-One“, polegający na dodaniu jedynki. Wygładzanie Laplace'a zapobiega zerowaniu prawdopodobieństwa dla zdarzeń, które nie zostały zaobserwowane podczas procesu nauczania i jednocześnie eliminuje możliwość osiągnięcia zdarzeń pewnych.

$$a'_{ij} = \frac{\mathcal{L}(S_i \rightarrow S_j) + 1}{\sum_k \mathcal{L}(S_i \rightarrow S_k) + N}, \quad b'_i(k) = \frac{\mathcal{L}(S_j \rightsquigarrow V_k) + 1}{\sum_l \mathcal{L}(S_j \rightsquigarrow V_l) + M}. \quad (2.62)$$

Absolutne dyskutowanie

Wygładzanie przy wykorzystaniu masy prawdopodobieństwa uzyskanej metodą największej wiarygodności dokonywane jest przez algorytm absolutnego dyskutowania. Z każdego niezerowego prawdopodobieństwa wystąpienia zdarzenia zabierana jest jego niewielka część i równo dystrybuowana na zdarzenia o zerowym prawdopodobieństwie wystąpienia. Wygładzenie dla estymacji prawdopodobieństw przejść i emisji wyrazić można wzorem:

$$a'_{ij} = \begin{cases} \frac{\mathcal{L}(S_i \rightarrow S_j) - \sigma_i}{\sum_k \mathcal{L}(S_i \rightarrow S_k)}, & \text{jeżeli } \mathcal{L}(S_i \rightarrow S_j) > 0; \\ \frac{\nu_{S(i)}^k \cdot \sigma_i}{N - \nu_{S(i)}}, & \text{w. p. r.} \end{cases} \quad b'_i(k) = \begin{cases} \frac{\mathcal{L}(S_j \rightsquigarrow V_k) - \sigma_j}{\sum_l \mathcal{L}(S_j \rightsquigarrow V_l)}, & \text{jeżeli } \mathcal{L}(S_j \rightsquigarrow V_k) > 0; \\ \frac{\nu_{V(j)}^l \cdot \sigma_j}{M - \nu_{V(j)}}, & \text{w. p. r.} \end{cases} \quad (2.63)$$

Współczynniki ν_S i ν_V określają odpowiednio liczbę przejść oraz liczbę emisji z niezerowym prawdopodobieństwem (czyli niezerowe wartości z macierzy przejść i emisji) dla danego stanu. Współczynnik dyskutowania σ wynika z liczby niezerowych prawdopodobieństw i wynosi $\frac{1}{1+\nu}$.

Promowanie poprawnych i łagodzenie błędnych (trening Viterbiego)

Zakładając, że pierwotnym źródłem badanej sekwencji obserwacji jest pewien proces Markowa oraz że sekwencja obserwacji pozbawiona jest anomalii mogących negatywnie wpłynąć na wynik dekodowania, to posiadając model HMM nauczony na tej sekwencji, powinno być możliwe zdekodowanie ścieżki stanów – odpowiadającej ścieżce, na podstawie której model Markowa został nauczony – z bardzo wysoką dokładnością (blisko 100%). Innymi słowy, prawidłowo nauczony model powinien być w stanie odwzorować sekwencję uczącą. W przeciwnym wypadku należy założyć, że uczenie modelu nie powiodło się. Przyczyną tej sytuacji może być niedostatek danych uczących, wadliwie dobrany rodzaj obserwacji lub źle dobrany algorytm uczenia.

Poprawę tej sytuacji osiągnąć można wprowadzając wariant treningu Viterbiego (Lember & Koloydenko 2008). Znając prawidłową sekwencję stanów dla badanej sekwencji obserwacji oraz najbardziej prawdopodobną sekwencję stanów otrzymaną w wyniku dekodowania modelu HMM algorytmem Viterbiego, wskazać można miejsca błędnie zdekodowanych stanów w ścieżce. Na tej podstawie można przeprowadzić dopasowanie modelu w sposób iteracyjny np. przez wypromowanie poprawnych tudzież łagodzenie błędnych parametrów w modelu. Jeżeli dana obserwacja w danym momencie czasowym t doprowadziła do błędnego stanu, to przy kolejnej iteracji estymacji prawdopodobieństwo tej obserwacji pod warunkiem błędnego stanu zostanie pomniejszone. Analogiczna sytuacja występuje przy ponownej estymacji macierzy przejść. Przy określaniu licznosci wystąpień danego zdarzenia (w celu określenia jego częstości) zwyczajowo suma wystąpień zwiększana jest o 1.00 za każdym razem kiedy dane zdarzenie zostało wykryte. Idea promowania lub łagodzenia wybranych parametrów polega na odpowiedniej modyfikacji (np. o 10%) wartości sumowanej, takiej że dla promowania 1.10, a dla łagodzenia 0.90. Wartość odchylenia parametrów powinna być relatywnie mała, pozwalając na dokonanie licznych iteracji i stopniowe dopasowywanie modelu. Należy ustalić granicę iteracji np. na podstawie procentowej zawartości błędnie zdekodowanych stanów w ścieżce. W wyniku takiego treningu Viterbiego wymuszone zostanie dopasowanie modelu względem danych, co jest jedynie lokalnie optymalne, a więc kolejne iteracje nie muszą prowadzić do osiągnięcia globalnie optymalnego rozwiązania. Dlatego też wynik każdej kolejnej iteracji należy śledzić, a trening przerwać, jeżeli dalsza poprawa modelu nie występuje.

2.7 Aplikacja HMM-Toolbox

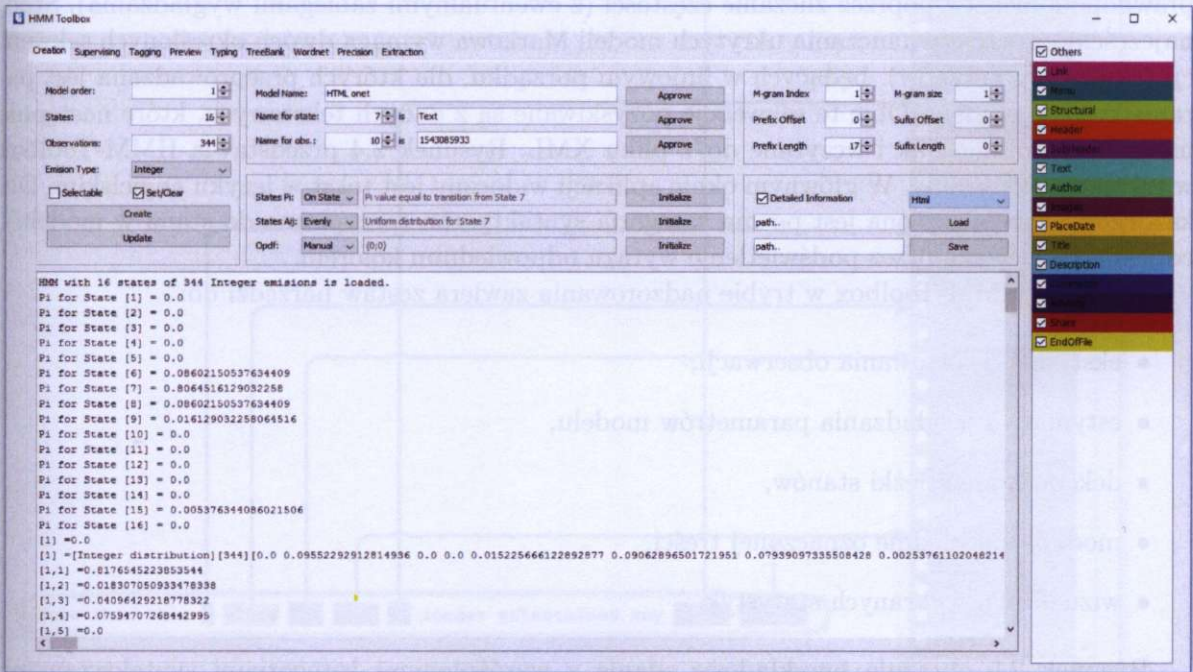
Głównym celem niniejszej pracy jest zaprojektowanie takich ukrytych modeli Markowa, które pozwolą na analizę danych tekstowych poprzez algorytmy programowania dynamicznego, realizowane sprzętowo. Stąd też jednym z produktów końcowych niniejszej rozprawy stał się zestaw opracowanych modeli Markowa, przygotowanych w formacie XML z wykorzystaniem autorskiej aplikacji. Wyprodukowanie pełnowartościowego modelu wymaga pewnej infrastruktury informatycznej, pozwalającej na tworzenie, zarządzanie, nauczanie, weryfikacje i wizualizacje modeli. Ze względu na różnorodność realizowanych celów szczegółowych, którymi są:

- pozyskiwanie istotnych danych tekstowych ze źródeł HTML,
- oznaczanie tekstu częściami mowy i częściami zdania,
- opis istotnych informacji w tekście na różnych poziomach semantyki leksykalnej,

na potrzeby prac badawczych utworzona została w pełni samodzielna aplikacja HMM-Toolbox. Aplikacja ta zawiera niezbędne zestawy narzędzi programistycznych w postaci przygotowanych funkcji i algorytmów oraz graficzny interfejs użytkownika usprawniający realizację poszczególnych zadań projektowych. Do obsługi modeli HMM aplikacja wykorzystuje biblioteki JAHMM (Francois 2006). Ze względu na operacje na obszernych strukturach baz danych (takich jak baza WordNet, sparsowane banki drzew frazowych i zależnościowych, źródła HTML) aplikacja ma dość wysokie zapotrzebowanie na pamięć roboczą. Zalecane parametry sprzętowe (PC) to: system operacyjny: 64-bitowy Windows 10; procesor: Core i3 lub Phenom II X4, lub lepszy; pamięć RAM: 4 GB; monitor: HD 1920 x 1080.

2.7.1 Tworzenie i zarządzanie modelami

Aplikacja HMM-Toolbox zapewnia obsługę standardowym ukrytym modelom Markowa, dla których zarówno przestrzeń stanów, jak i zbiór obserwacji są dyskretne. Struktura tworzonego modelu (tj. liczba stanów i obserwacji) jest w pełni konfigurowalna. Dodatkowe informacje opisujące model (takie jak nazwa i typ modelu, nazwy stanów i obserwacji) również są dostępne i edytowalne z poziomu GUI (Graficznego Interfejsu Użytkownika), co pokazuje Rysunek 2.3.



Rysunek 2.3: Przykładowa zawartość zakładki „Creation“ do tworzenia nowych modeli HMM w aplikacji HMM-Toolbox.
(Źródło: opracowanie własne).

Utworzony model domyślnie posiada jednostajne rozkłady prawdopodobieństw, niemniej w zależności od potrzeb rozkłady te mogą być ręcznie modyfikowane dla każdego ze stanów. Ponadto HMM-Toolbox obsługuje n -gramowy konfigurator obserwacji pozwalający na zdefiniowanie długości afiksów i prefiksów oraz rzędu gramowości. Przechowywanie modeli w HMM-Toolbox odbywa się za pomocą reprezentacji XML, a skonfigurowany model można zapisać i odczytać w strukturyzowany sposób do/z pliku XML.

Kolejne zakładki aplikacji HMM-Toolbox prezentowane na dalszych rysunkach odpowiadają typowemu przebiegowi pracy użytkownika z aplikacją:

- tworzenie modelu (zakładka *Creation* – Rysunek 2.3),
- uczenie nadzorowane modelu (zakładka *Supervising* – Rysunek 2.4),
- graficzna wizualizacja struktury drzewa (zakładka *Treebank* – Rysunek 2.6),
- zbiorczego oznaczania treści (zakładka *Tagging* – Rysunek 3.6),
- podgląd pozyskiwanej informacji (zakładka *Supervising* – Rysunek 3.8),
- wpisywanie i rozpoznawanie nowego tekstu (zakładka *Typing* – Rysunek 4.16),
- opis informacji na różnych poziomach semantyki (zakładka *Wordnet* – Rysunek 5.3),
- redukcja reprezentacji liczbowej (zakładka *Precision* – Rysunek 6.2).

Ponadto w zakładce *Extraction* znajdują się przydatne funkcje importu, eksportu i konwersji różnych struktur danych wykorzystanych podczas realizacji prac badawczych.

2.7.2 Uczenie modeli HMM na podstawie zbiorów danych

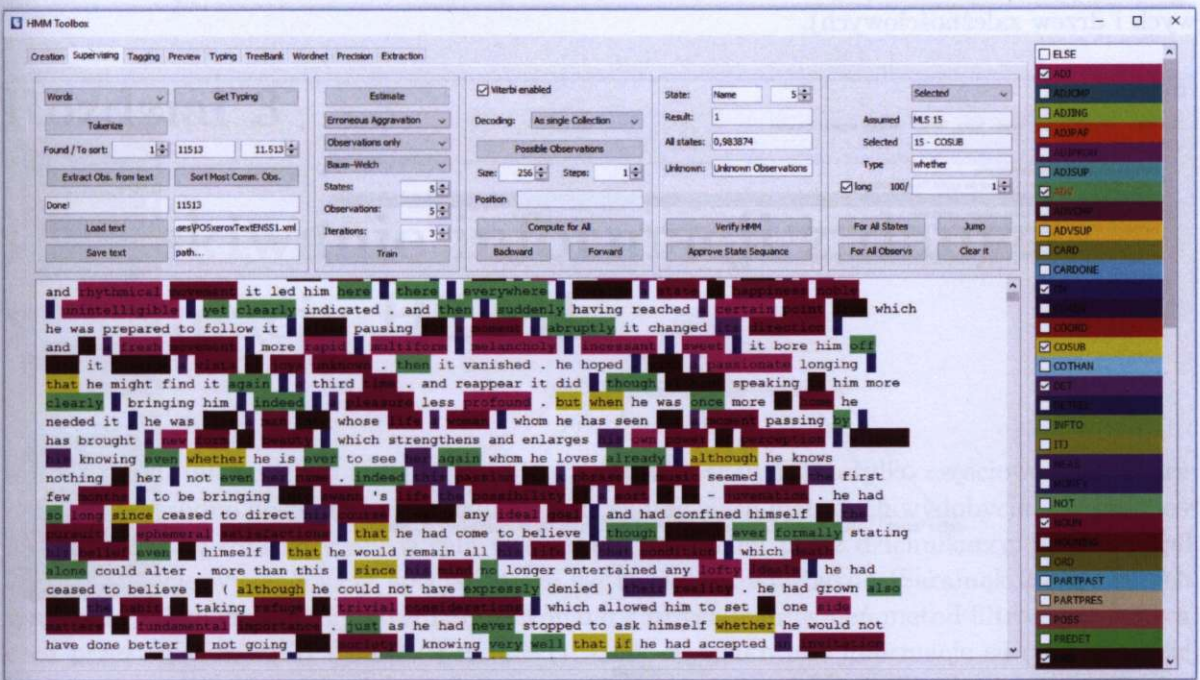
Zgodnie z wcześniejszymi uwagami uczenie modelu HMM może być przeprowadzane w sposób nadzorowany. Do tego celu potrzebny jest zbiór sekwencji obserwacji wraz z towarzyszącymi im ścieżkami stanów. Podstawowe uczenie sprowadza się wówczas do estymowania rozkładów prawdopodobieństw poprzez zliczanie częstości (z ewentualnymi zabiegami wygładzania). Stąd, najczęściej procedura nauczania ukrytych modeli Markowa wymaga dwóch określonych sekwencji (obserwacji i stanów), będących w liniowym porządku, dla których przeprowadzana jest parametryzacja modelu. Obie te sekwencje pozyskiwane są z danych tekstowych, które następnie mogą być zapisane, jak i wczytane do/z pliku XML. Rysunek 2.4 przedstawia HMM-Toolbox w trybie nadzorowania. W głównym oknie aplikacji widoczny jest tekst w języku angielskim. Dla każdego wyrazu określona jest pewna kategoria syntaktyczna (przypisana do stanu w modelu), co jest zobrazowane przez podświetlenie wyrazu odpowiednim kolorem.

Aplikacja HMM-Toolbox w trybie nadzorowania zawiera zestaw narzędzi do:

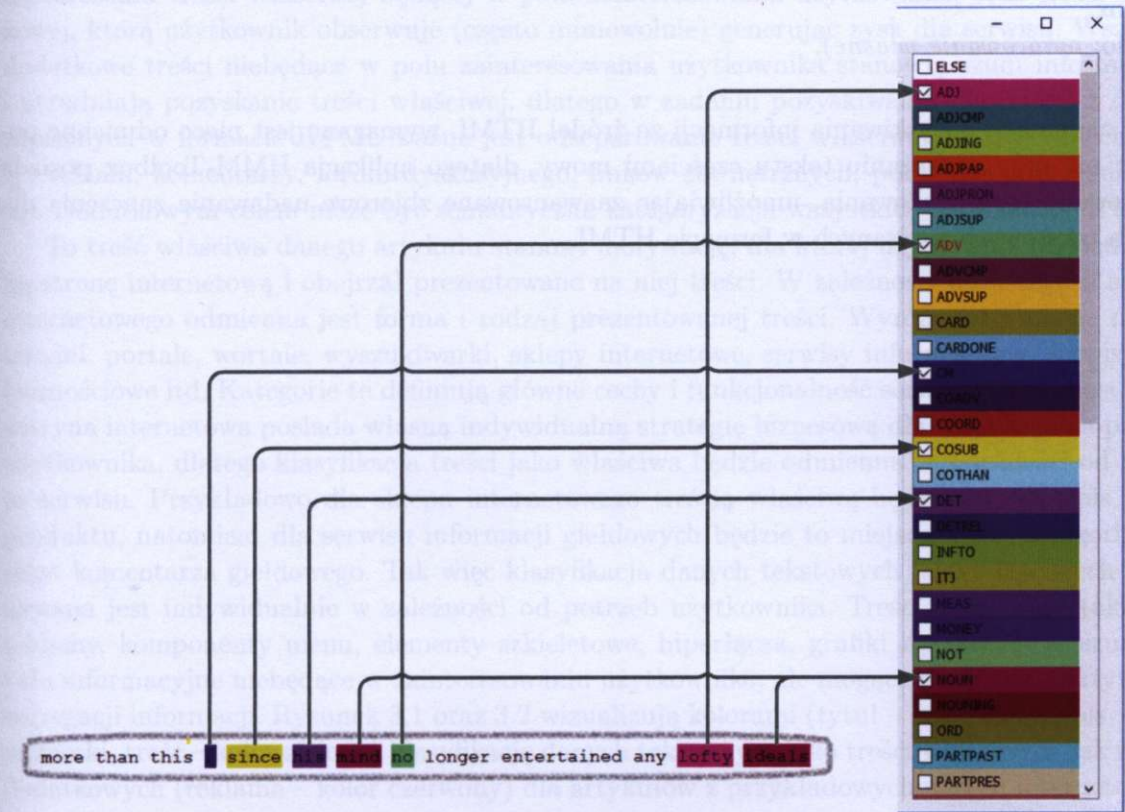
- ekstrakcji i sortowania obserwacji,
- estymacji i wygładzania parametrów modelu,
- dekodowania ścieżki stanów,
- modyfikacji błędnie oznaczonej treści,
- wizualizacji wybranych statystyk.

Rysunek 2.5 obrazuje przykładowe zdanie z wyróżnionymi kategoriami syntaktycznymi. Strzałkami przedstawiono kategorie reprezentowane przez dany wyraz. Podświetlanie wyrazu kolorem pozwala na szybką ocenę prawidłowego przypisania kategorii do wyrazu. Nieprawidłowo przypisane kategorie można poprawić ręcznie lub przez metody douczania modelu. Aplikacja

HMM-Toolbox posiada dodatkowy tryb podświetlania dla wyrazów błędnie oznaczonych oraz tryb podświetlania jedynie wybranych kategorii.



Rysunek 2.4: Nauczanie i weryfikacja modelu w HMM-Toolbox.
(Źródło: opracowanie własne).



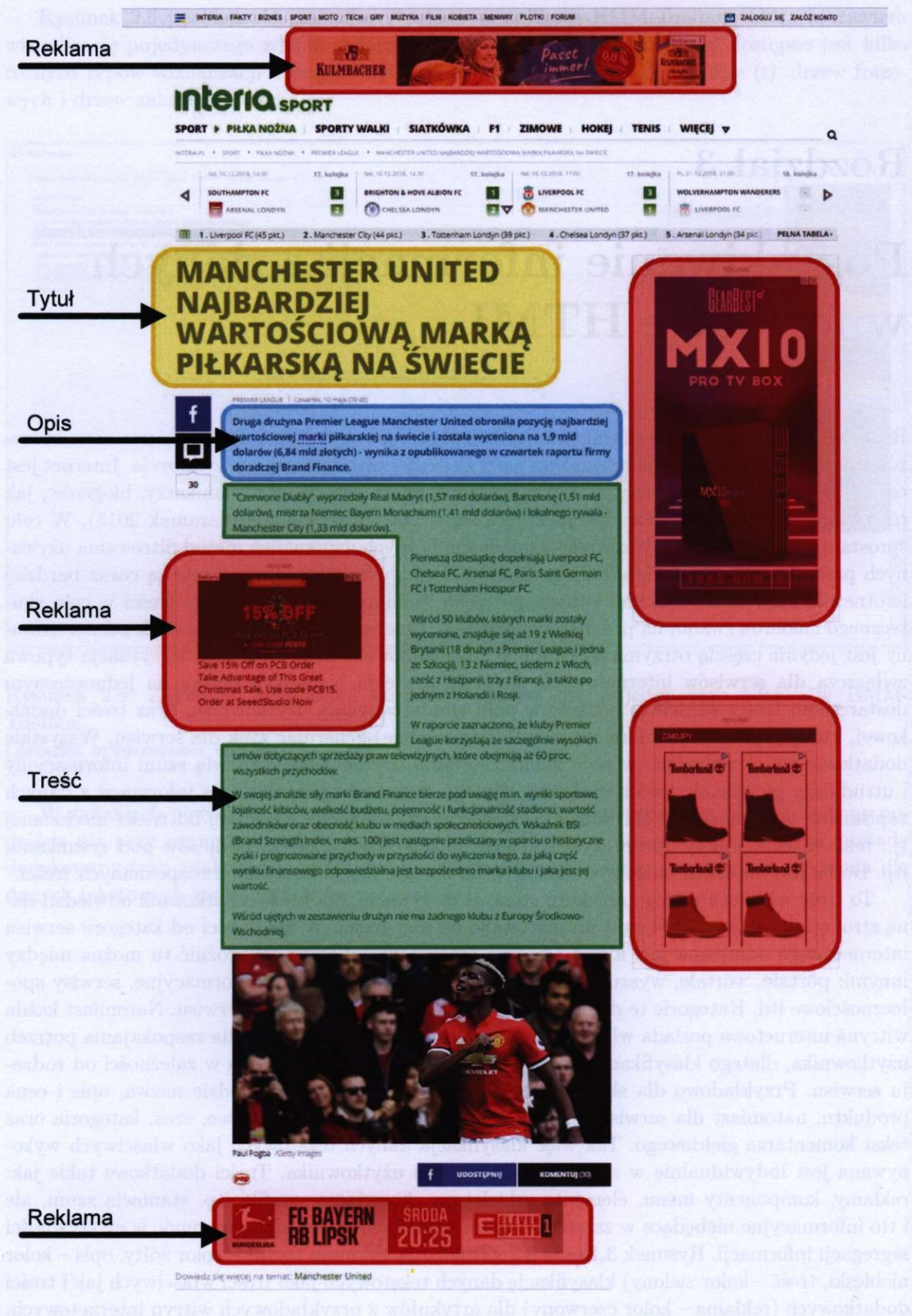
Rysunek 2.5: Przykład zdania w HMM-Toolbox z przypisanymi kategoriami syntaktycznymi do poszczególnych wyrazów.
(Źródło: opracowanie własne).

Rozdział 3

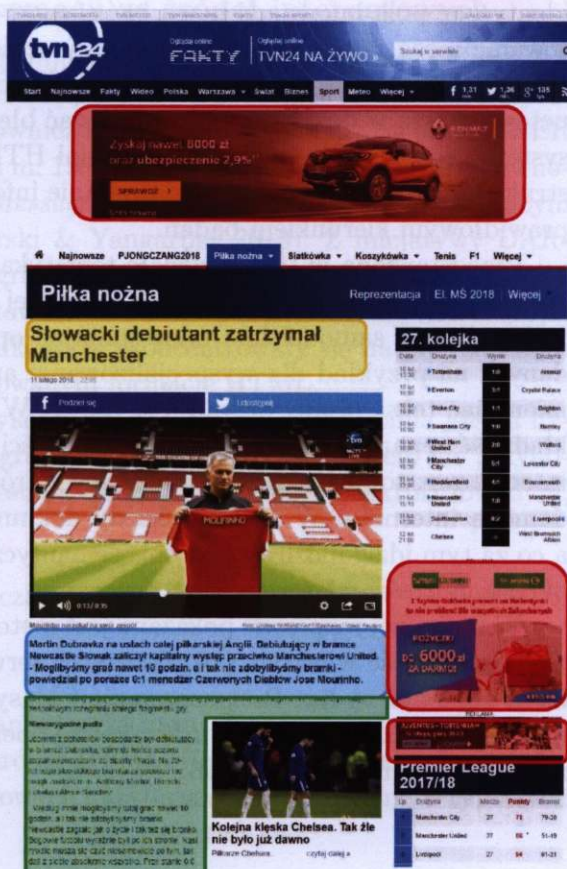
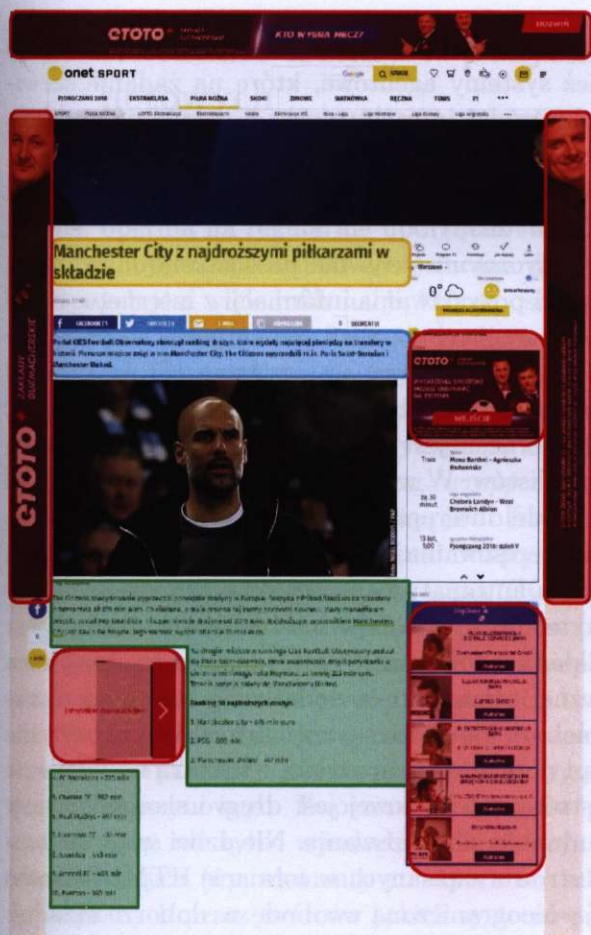
Pozyskiwanie informacji z danych w formacie HTML

Różnego typu informacje o charakterze nieustrukturyzowanym lub tylko częściowo ustrukturyzowanym są trudno dostępne ze względu na brak skutecznych metod ich wydobycia. Internet jest coraz częściej wykorzystywany w celu pozyskania informacji przez dziennikarzy, blogerów, jak również agencje wywiadowcze specjalizujące się w białym wywiadzie (Saramak 2015). W celu sprostania wymaganiom użytkowników dokonano licznych usprawnień metod filtrowania używanych przez wyszukiwarki (Madhu i in. 2011). Choć wyszukiwane informacje są coraz bardziej istotne dla zapytania, to nadal istnieje potrzeba semantycznej identyfikacji treści w celu skutecznego nakierowywania na pożądane informacje. Przeważnie jednak użytkownik zainteresowany jest jedynie częścią otrzymanej treści, a „skonsumować” musi całą. Jest to sytuacja typowa zwłaszcza dla serwisów internetowych, których strategia biznesowa polega na jednoczesnym dostarczeniu treści właściwej będącej w polu zainteresowania użytkownika, oraz treści dodatkowej, którą użytkownik obserwuje (często mimowolnie) generując zysk dla serwisu. Wszystkie dodatkowe treści niebędące w polu zainteresowania użytkownika stanowią szum informacyjny i utrudniają pozyskanie treści właściwej, dlatego w zadaniu pozyskiwania informacji z danych zapisanych w formacie HTML ważne jest odseparowanie treści właściwej od treści niechcianej tj.: reklam, komentarzy, forum dyskusyjnego, linków zewnętrznych, podpisów pod rysunkami, itp. Dodatkowym celem może być semantyczna kategoryzacja wszystkich rozpoznanych treści.

To treść właściwa danego artykułu stanowi motywację, dla której użytkownik odwiedził daną stronę internetową i obejrzał prezentowane na niej treści. W zależności od kategorii serwisu internetowego odmienna jest forma i rodzaj prezentowanej treści. Wyróżnić tu można między innymi: portale, wortale, wyszukiwarki, sklepy internetowe, serwisy informacyjne, serwisy społecznościowe itd. Kategorie te definiują główne cechy i funkcjonalność serwisu. Natomiast każda witryna internetowa posiada własną indywidualną strategię biznesową dla zaspokajania potrzeb użytkownika, dlatego klasyfikacja treści jako właściwa będzie odmienna w zależności od rodzaju serwisu. Przykładowo dla sklepu internetowego treścią właściwą będzie nazwa, opis i cena produktu, natomiast dla serwisu informacji giełdowych będzie to miejsce, czas, kategoria oraz tekst komentarza giełdowego. Tak więc klasyfikacja danych tekstowych jako właściwych wykonywana jest indywidualnie w zależności od potrzeb użytkownika. Treści dodatkowe takie jak: reklamy, komponenty menu, elementy szkieletowe, hiperłącza, grafiki itp. stanowią szum, ale i tło informacyjne niebędące w zainteresowaniu użytkownika, ale mogące pomóc w efektywności segregacji informacji. Rysunek 3.1 oraz 3.2 wizualizują kolorami (tytuł – kolor żółty, opis – kolor niebieski, treść – kolor zielony) klasyfikację danych tekstowych jako treści właściwych jak i treści dodatkowych (reklama – kolor czerwony) dla artykułów z przykładowych witryn internetowych.



Rysunek 3.1: Przykładowa witryna internetowa z oznaczoną treścią właściwą artykułu oraz treścią dodatkową.
(Źródło: opracowanie własne).



Rysunek 3.2: Przykładowe witryny internetowe z oznaczoną treścią właściwą artykułów oraz treścią dodatkową.
(Źródło: opracowanie własne).

Należy pamiętać, że przeszukiwanie zasobów internetowych w celu pozyskania informacji to nie tylko domena ludzi. Zajmują się tym również systemy agentowe, które na żądanie serwisów lub użytkowników zbierają informacje z wielu źródeł internetowych, a następnie poddają je odpowiedniej konwersji i filtrowaniu tak, aby wydobyć informację będącą w polu zainteresowania (przykładowe rozwiązania pozyskiwania rekomendacji na podstawie źródeł stron serwisu informacji giełdowych przedstawiono w (Wiliński i in. 2015)).

Prace badawczo-rozwojowe nad sieciami semantycznymi aktywnie prowadzone m.in. przez konsorcjum W3C, mają na celu poprawę skuteczności pozyskiwania informacji z internetu. Z powodzeniem stosowane w obecnych serwisach informacyjnych są sieci semantyczne wspierające format RSS (ang. *Really Simple Syndication*, *Rich Site Summary*) oraz Atom (Stolz & Hepp 2013), specjalnie zaprojektowany do łatwej i szybkiej wymiany zawartości witryn internetowych. Wyszukiwanie, filtrowanie, zbieranie informacji w internecie jest czasochłonne i nierzadko wymaga zaangażowania wysoko wykwalifikowanych specjalistów. W większości przypadków specjaliści zaangażowani w zbieranie i analizę informacji ze źródeł internetowych mają dość jasne wyobrażenie, jakie są informacje i źródła w kręgu ich zainteresowania, stąd tematyczna kategoryzacja informacji tekstowych jest warunkiem wstępnym dla automatyzacji pozyskiwania wiedzy.

Jednakże ze względu na zróżnicowany charakter opisu zasobów internetowych trudno jest precyzyjnie zlokalizować pożądane informacje. Składnia formatu HTML pozwala na dużą swobodę w opisie informacji, nie posiada jednoznacznej semantyki, a dodatkowo kod zawiera zagnieżdżone skrypty (np. JavaScript). Treści internetowe tworzone są, więc tak, aby były czytelne dla człowieka, co nie musi iść w parze ze staranną i dobrze zorganizowaną strukturą na poziomie kodu źródłowego. To sprawia, że opis zasobów strony internetowej jest długi i skomplikowany do tego stopnia, że przydatna informacja jest trudna do zlokalizowania. Nie dziwi więc, że konwencjonalne metody wydobywania informacji ze stron z zapisanych w formacie HTML są mało skuteczne głównie dlatego, że twórcy stron mają nieograniczoną swobodę w doborze składni. Wynikiem tego jest różnorodność metod nadawania struktury danych na stronie – tak, że dwa identyczne pola informacji mogą być sformatowane w całkowicie różny sposób. Ponadto formatowanie może zawierać błędy, to znaczy część kodu może być nie w pełni zgodna z gramatyką HTML (brak tagów zamknięcia jest tego typowym przykładem). Dlatego też przeglądarki internetowe muszą w dużym stopniu tolerować błędy i luźną składnię formatu HTML. Reasumując – system pozyskiwania informacji z źródeł HTML również powinien wykazywać tolerancję co do struktury składniowej, stąd wydobywanie informacji w oparciu o znaczniki HTML zdaje się być prawidłowym kierunkiem badań.

Przedstawiona w tym rozdziale tematyka pozyskiwania informacji z źródeł HTML stanowi etap w analizie danych tekstowych, niemniej przedstawiona procedura może być równie dobrze wykorzystana samodzielnie udostępniając opis semantyczny na poziomie warstwy prezentacji danych na przykład dla wyspecjalizowanej aplikacji syntezy mowy (Tokuda i in. 2013) zamieniając tekst pisany artykułu w mowę. W tego typu aplikacjach wymagana jest wysoka dokładność separacji treści właściwej od treści niechcianej. Mimo, że analizowane są tu serwisy internetowe, to pozyskiwanie informacji z źródeł HTML ma szerokie zastosowanie również poza domeną internetu. W zasadzie każdy dokument może zostać przedstawiony w formie HTML, a co za tym idzie możliwy jest opis semantyczny danych na poziomie prezentacji, a tym samym określenie treści właściwej.

W niniejszej rozprawie pokazana jest metoda pozyskiwania informacji z serwisów informacyjnych, stąd treść właściwa pozyskiwana z serwisu dotyczyć będzie artykułu, tj. tematu, autora, daty, komentarza itp. Rozpoznawanie i klasyfikacja informacji dokonana została przy pomocy nauczonego ukrytego modelu Markowa, operującego na odpowiednio przygotowanych danych zapisanych w formacie HTML.

3.1 Aktualne trendy w pozyskiwaniu informacji ze źródeł internetowych

Dla bardziej efektywnego manipulowania danymi internetowymi niektórzy badacze zwrócili się do idei zaczerpniętych z dziedziny baz danych. Tradycyjne algorytmy przetwarzania baz danych operują na regularnie uporządkowanych źródłach informacji i dlatego takie technologie nie mogą być bezpośrednio zastosowane do wydobywania informacji z sieci. Jednakże w sieci dostępnych jest wiele informacji, które posiadają specyficzną strukturę (np. katalog elektroniczny, elektroniczne media, itd.), co zachęca do prowadzenia badań w zakresie dostępu do tych informacji, które nazywa się zwykle częściowo ustrukturyzowanymi.

Tradycyjne podejście zakłada pozyskiwanie danych z sieci WWW i przekształcenie ich do formy pośredniej, a następnie wykonanie klasyfikacji. Podejście to jest realizowane przy pomocy programów pośredniczących (ang. *wrapper*), które identyfikują wymagane informacje w dokumencie źródłowym i przekształcają je do formy pośredniej, na przykład XML. Zgodnie z przyjętą nomenklaturą taki program pośredniczący do wydobywania informacji z częściowo ustrukturyzowanych danych nosi nazwę „wrapper” (Kushmerick i in. 1997).

Ręczne tworzenie i rozwijanie programów pośredniczących jest czasochłonnym procesem (w szczególności ze względu na różnorodność i nieprawidłowości przetwarzanych informacji), dlatego różne grupy specjalistów w wielu ośrodkach badawczych pracują nad automatyzacją tego zadania. W tym celu dokonuje się analizy zbioru dokumentów, z których planowane jest pozyskiwanie wybranych informacji. Na tej podstawie przygotowuje się przykłady szkoleniowe dla algorytmów uczenia maszynowego. Tworzenie programów pośrednich jest więc złożonym zadaniem, ponieważ wymaga nie tylko wyodrębnienia rodzaju danych, ale także związanych z nimi informacji semantycznych (np. elementy strukturalne, reklama, menu, itp.). Jedną z głównych trudności napotykaną przez twórców takich systemów jest nieregularność przetwarzanych informacji.

3.1.1 Rozwój technik pozyskiwania informacji ze źródeł internetowych

Badania w dziedzinie ekstrakcji informacji (ang. *Information Extraction*) sięgają połowy XX wieku, jednak pierwsze systemy operujące na dużą skalę pojawiły się w latach siedemdziesiątych, np. Lockheed Dialog System. Niedługo powstały również komercyjne systemy takie jak JASPER - wydobywający ważne dane finansowe (Andersen i in. 1992). Do rozwoju problematyki z pewnością przyczyniła się seria konferencji MUC (ang. *Message Understanding Conference*) w dużym stopniu wspierana przez agencję DARPA (Piskorski & Yangarber 2013). Z inicjatywy DARPA w latach 90-tych prowadzony był również program TIPSTER, którego celem była poprawa maszynowego przetwarzania i rozumienia tekstu głównie w celach wywiadowczych. Wraz z pojawieniem się i popularyzacją internetu badania naukowców skoncentrowały się na pozyskiwaniu informacji zagnieżdżonej w częściowo ustrukturyzowanym formacie HTML.

Tradycyjny system pozyskiwania informacji wykorzystuje technikę przetwarzania języka naturalnego w oparciu o składniowe i semantyczne ograniczenia, jednakże taki system nie przynosi pożądanych wyników. Dokument internetowy jest przekazem częściowo ustrukturyzowanym, który posiada dwie istotne cechy:

1. zawiera dużą ilość tekstu oraz liczne hiperłącza;
2. tekst rzadko operuje pełnymi zdaniami.

Z powodu tych dwóch cech technika przetwarzania języka naturalnego nie jest w pełni stosowana do pozyskiwania informacji bezpośrednio z źródeł HTML. Niniejszy rozdział opisuje, więc metody pozyskiwania informacji ze źródeł internetowych i kryteria oceny stosowane w typowych rozwiązaniach.

3.1.2 Sposób pozyskiwania informacji ze źródeł internetowych

Zestaw procedur i filtrów wykorzystywany przez program pośredniczący często oparty jest na systemie ekspertyzowym wykorzystującym inżynierię wiedzy, gdzie eksperci dokonują konfiguracji reguł klasyfikacji (przykładowy system klasyfikacji na podstawie reguł decyzyjnych pokazano w (Pietras 2013)). Jakość określonych reguł jest zależna od umiejętności specjalistów w danej dziedzinie. Niestety strony internetowe są zróżnicowane w formie i stale aktualizowane, co sprawia, że ręczna długotrwała konfiguracja jest uciążliwa, a wręcz czasem niemożliwa do przeprowadzenia. Samo oznakowanie nawet dużej liczby dokumentów przez człowieka jest zwykle zadaniem mniej czasochłonnym niż opracowanie skutecznych filtrów i reguł (które miałyby skutecznie wyłuskiwać treść) do tychże dokumentów. Aktywność badawcza w dziedzinie przetwarzania informacji częściowo strukturalnych doprowadziła do powstania dużej liczby alternatywnych narzędzi wykorzystywanych do tworzenia programów pośredniczących. Proponowane podejścia wydobywania danych ze stron internetowych wykorzystują techniki zapożyczone z dziedzin takich jak: przetwarzanie języka naturalnego, indukcja reguł gramatyki, uczenie maszynowe, technologie baz danych, oraz ontologii. Istnieje wiele innych podejść do problemu wydobywania informacji częściowo ustrukturyzowanej ze źródeł internetowych. Korzystają one z drzew logiki pierwszego i drugiego rzędu (Crescenzi & Mecca 2004, Gottlob & Koch 2001, Michael Benedikt & Neven 2007, Courcelle 2006). Techniki te są obiecujące, jednakże skuteczna ich realizacja wymaga opracowania nowych sposobów prezentacji i przechowywania dokumentów w postaci struktury drzewa ze zintegrowanym językiem zapytań, co wymaga dużego wysiłku w fazie rozwojowej. W rezultacie zapewniają różne możliwości, więc surowe kryteria porównania nie dają jednoznacznych wniosków.

Dzięki wykorzystaniu uczenia maszynowego reguły klasyfikacji dla programów pośredniczących dobierane są automatycznie. Algorytm uczenia bazując na odpowiednio otagowanej treści tworzy model reprezentacji wiedzy. Źródła stron internetowych poddawane są wstępnemu przetwarzaniu w celu pozyskania odpowiedniej reprezentacji danych wymaganej przez algorytm. Na podstawie algorytmu nauczonego maszynowo dokonywana jest klasyfikacja danych. W wyniku tego procesu otrzymywana jest odpowiednio skategoryzowana informacja.

3.1.3 Reprezentacja danych uzyskanych z formatu HTML

Dla celów pozyskiwania informacji z danych zapisanych w formacie HTML pod uwagę mogą być brane dowolne oznaczenia i symbole tekstowe zawarte w kodzie HTML. Wyraźnie rozróżnialne i odseparowane wybrane sekwencje symboli, znaków czy liter noszą nazwę tokenów. Tokeny wstępnie identyfikują dane tekstowe i w zależności od potrzeb dalszego przetwarzania mogą być grupowane na różne sposoby.

Koncepcja sekwencji tokenów

Sekwencja tokenów z źródeł zapisanych w formacie HTML tworzona jest przez separację tekstu zapisanego w języku naturalnym od znaczników HTML. Pojedynczy token określa cechy charakterystyczne reprezentowanego przez siebie wycinka danych tekstowych.

Przykładowy fragment kodu HTML:

```
1 <h1> Wybory 2017 </h1>
```

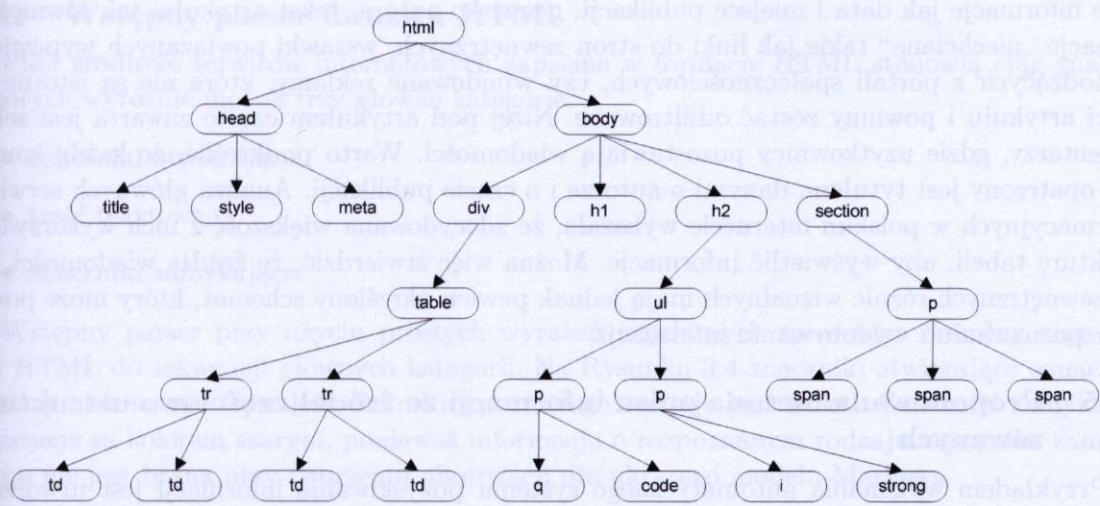
może być zamieniony na ciąg tokenów opisujących kolejne wyrażenia:

1. token (type = html_open, tag = h1),
2. token (type = text, txt = "Wybory 2017"),
3. token (type = html_close, tag = h1).

Ze względu na szczególne znaczenie funkcjonalne w tokenie wyróżniona jest informacja, czy dany znacznik jest znacznikiem otwierającym, czy też zamykającym. Ponadto przestrzeń między znacznikami może być określona jako zawierająca tekst napisany w języku naturalnym lub też inny ciąg znaków. Ciąg tokenów nie posiada jednak struktury hierarchicznej i wszystkie jego elementy rozpatrywane są na jednym poziomie.

Koncepcja drzewa DOM

Ze struktury plików formatu HTML można wygenerować obiektowy model dokumentu DOM (ang. *Document Object Model*) i na tej podstawie można użyć interfejsu aplikacji W3C, który pozwala na dostęp i łatwe manipulowanie elementami struktury DOM. Koncepcja drzewa HTML zaproponowana w pracy (Dipasquo 1998), ze względu na przejrzystość struktury (patrz Rysunek 3.3) jest często wykorzystywana w algorytmach pozyskiwania informacji, jak to zaprezentowano w pracy (Embley i in. 1999), gdzie użyta została struktura DOM oraz uczenie indukcyjne do generowania reguł klasyfikacji. Przeszukiwanie drzewa, jak i stosowanie reguł przedstawione zostało w pracy (Embley i in. 1998).



Rysunek 3.3: Przykład hierarchii w obiektowym modelu dokumentu HTML.
(Źródło: opracowane na podstawie (Embley i in. 1998)).

Korzeniem drzewa DOM dla strony HTML jest tag `<html>`. W drzewie DOM zachowana jest hierarchia znaczników HTML odwzorowana za pomocą kolejnych coraz do głębiej osadzonych węzłów drzewa. Liście drzewa reprezentują pola tekstowe. Dla książkowo napisanego pliku w czystym formacie HTML reprezentacja DOM powinna w pełni odwzorować hierarchie obiektów. Jednakże język HTML nie posiada standardowej nawigacji ani standardowej struktury, a pliki źródłowe nie są jednolicie zorganizowane, posiadają wiele elementów dodatkowych takich jak zagnieżdżone skrypty. Dlatego też można założyć, że aby uzyskać informację z dokumentów HTML – należy zaakceptować niejednolitość struktury oraz różnorodność składni.

3.1.4 Typowa struktura stron internetowych

Różnorodność witryn i stron internetowych jest wynikiem kreatywnej pracy ich projektantów. Jediną rzeczą, która je łączy jest język HTML, który definiuje wygląd zasobów internetu, ale nie opis jego zawartości. Hipertekstowy język znaczników (ang. *Hypertext Markup Language*) wykorzystywany jest powszechnie przez serwisy sieciowe do modelowania struktury jak i wyglądu przekazywanej informacji. Format HTML wykorzystuje znaczniki opisowe (zwane tagami), aby wskazać różne elementy zawartości dokumentu. Znaczniki mają zwykle dwie role: pierwszorzędną – zdefiniować strukturę dokumentu, dzięki czemu przeglądarka może go wyświetlić

i drugorzędną – umożliwiającą (np. systemom agentowym) przeszukiwanie treści tak, aby nakierować na główny obszar dokumentu. Format HTML5 jako piąta odsłona języka wzbogacony został o nowe znaczniki, w tym semantyczne rozwiązania XML znane z formatu XHTML. Dzięki standaryzacji, informacja zawarta w formacie HTML5 jest prawidłowo prezentowana przez przeglądarki na dowolnym sprzęcie komputerowym. W zamyśle format HTML pozwala na prezentację informacji niezależnie od jej rodzaju, formy czy języka, a końcowym odbiorcą informacji jest człowiek. Dzięki rozbudowanej warstwie prezentacji możliwe jest przedstawienie informacji na urządzeniu użytkownika, który ją interpretuje i konsumuje zadane treści. Wizualizacja informacji w postaci tekstu, elementów grafiki, obiektów multimedialnych takich jak obrazy i filmy wzbogaca przekaz i sprawia, że treść może być subiektywnie przyjemniejsza w odbiorze.

Pomimo braku podobieństwa wizualnego pomiędzy różnymi stronami internetowymi, wyróżnić można pewne cechy grupujące tematycznie witryny internetowe, na przykład jako portale informacyjne, portale społecznościowe czy wyszukiwarki. Analizując przykładowe portale informacyjne można zauważyć, że większość z nich stosuje pewien ogólnie przyjęty schemat prezentacji artykułu. Wizualnie wyróżnić można tytuł artykułu zwykle pisany większą czcionką w górnej części strony, opis artykułu zwykle znajdujący się w bliskim sąsiedztwie pod tytułem, oraz treść artykułu jako znaczny blok tekstu znajdujący się pod opisem. Blok ten zawierać może takie informacje jak data i miejsce publikacji, nazwisko autora, tekst artykułu; jak również informacje „niechciane” takie jak linki do stron zewnętrznych, wstawki powiązanych wypowiedzi pochodzących z portali społecznościowych, czy wbudowane reklamy, które nie są istotne dla treści artykułu i powinny zostać odfiltrowane. Niżej pod artykułem często zawarta jest sekcja komentarzy, gdzie użytkownicy pozostawiają wiadomości. Warto podkreślić, że każdy komentarz opatrzony jest tytułem, danymi o autorze i o czasie publikacji. Analiza głównych serwisów informacyjnych w polskim internecie wykazała, że zdecydowana większość z nich wykorzystuje strukturę tabeli, aby wyświetlić informacje. Można więc stwierdzić, że źródła wiadomości, mimo zewnętrznych różnic wizualnych mają jednak pewien określony schemat, który może pomóc w rozpoznawaniu i wydobywaniu informacji.

3.1.5 Proponowana metoda opisu informacji ze źródeł częściowo ustrukturyzowanych

Przykładem wykonania automatycznego systemu pozyskiwania informacji jest utworzenie programu pośredniczącego korzystającego z osobnych modeli dedykowanych dla każdego źródła informacji. W tej metodzie realizacji dla każdego źródła informacji trzeba utworzyć osobny model zapewniający wydobywanie informacji, a logika programu pośredniczącego decyduje, który model najtrafniej według przyjętego kryterium wydobędzie informacje ze strony internetowej. Można wyobrazić sobie np. oprogramowanie, które będzie dysponować zestawem poszczególnych modeli HMM np. dla polskich stron informacyjnych: *onet.pl*, *wp.pl*, *tvn24.pl*, itp. Jako dalszy rozwój systemu można sobie wyobrazić rozrost bazy modeli pozwalający na rozpoznawanie szerokiej gamy różnych witryn internetowych.

Proponowanym modelem dokonującym klasyfikacji danych tekstowych zapisanych w formacie HTML jest ukryty model Markowa, który na podstawie sekwencji obserwacji przypisuje do danych odpowiednie kategorie. Ponadto, dla danego ukrytego modelu Markowa możliwe jest określenie jakie byłoby prawdopodobieństwo wygenerowania badanej ścieżki obserwacji. Prawdopodobieństwo to można porównywać między poszczególnymi modelami w celu znalezienia modelu dokonującego najtrafniejszej kategoryzacji danych z formatu HTML. Warto podkreślić, że zaprojektowany system informatyczny, ma pozwolić użytkownikowi bez specjalnego przeszkolenia (nie będącemu programistą) na odpowiednie oznakowanie treści witryny internetowej i na tej podstawie zostanie wygenerowany model.

Przykładem realizacji takiego edytora wizualnego jest zakładka „Tagging” w aplikacji HMM-Toolbox. Treści są w niej znakowane według wcześniej zdefiniowanej przestrzeni stanów modelu (kategorii semantycznych). Główną zaletą edytora wizualnego jest sprawne oznakowywanie treści. Czynność ta wykonywana jest ręcznie (ewentualnie półautomatycznie), stąd zadanie to jest

czasochłonne i to od ergonomiczności i przyjazności edytora zależy efektywność procesu oznaczania. Zagadnienia ergonomicznego projektowania interfejsu użytkownika – choć są poza zakresem tematycznym niniejszej rozprawy – stanowią ważny aspekt tworzenia aplikacji. Zostały one obszernie opisane w pracy (Ghaoui 2005). Tak więc – przy korzystaniu z graficznego edytora – możliwe jest określenie głównych etapów przygotowania danych z nowego źródła internetowego w następujący sposób; najpierw należy określić kategorie wydobywanych informacji (tj. tytuł, treść, data, autor, itd.), następnie należy załadować przykładowe źródła HTML dla wybranych stron internetowych. Na tej podstawie ekspert dokonuje przypisania kategorii znaczeniowych do elementów strony internetowej. Ostatnim krokiem jest już tylko nauczanie modelu. Prawidłowo nauczony model może być od razu stosowany przez „wrapper“, a w razie potrzeby może zostać douczony. Douczanie nie tylko wpływa na polepszenie skuteczności klasyfikacji, ale przede wszystkim pozwala na dostosowanie modelu w przypadku aktualizacji lub modyfikacji kodu HTML strony internetowej.

3.2 Dane w formacie HTML jako sekwencja obserwacji dla ukrytego modelu Markowa

3.2.1 Wstępny parser formatu HTML

Pliki źródłowe serwisów internetowych zapisane w formacie HTML stanowią ciąg znaków, z których wyróżnić można trzy główne kategorie:

- znaczniki otwierające,
- treść tekstowa,
- znaczniki zamykające.

Wstępny parser przy użyciu prostych wyrażeń regularnych sprowadza dowolny ciąg znaków HTML do sekwencji głównych kategorii. Na Rysunku 3.4 znaczniki otwierające oznaczone są różnymi kolorami (w zależności od rozpoznanego rodzaju znacznika). Znaczniki zamykające oznaczone są kolorem szarym, ponieważ informacja o rozpoznanym rodzaju znacznika zamykającego nie jest brana przy tworzeniu obserwacji dla ukrytego modelu Markowa.

```
<span class="follow" data-favapi-uid="420b7bbc-f2e2-4f23-bb7c-ad9a5f4f117f" data-favapi-itemtype="topic">
<span>
<div class="source">

<meta itemprop="datePublished" content="2016-02-11 21:11:21+0100"/>
<div class="datePublished">
11 lut, 21:11
</div>
<div id="mainTitle" class="mainTitle">
<h1 itemprop="name headline">
<meta itemprop="http://purl.org/dc/terms/identifier" content="a1a145be-bb54-75ce09d" />
Problemy sektora turystyki w Egipcie
</h1>
<div class="authorSourceProfile authorProfile" itemscope itemType="http://schema.org/Person" itemprop="author">
<div class="logo">

</div>
<div class="profileDetails">
<header>
<a href="http://wiadomosci.onet.pl/autorzy/jacek-gadek">
<span class="name" itemprop="name">
Jacek Gadek
</span>
```

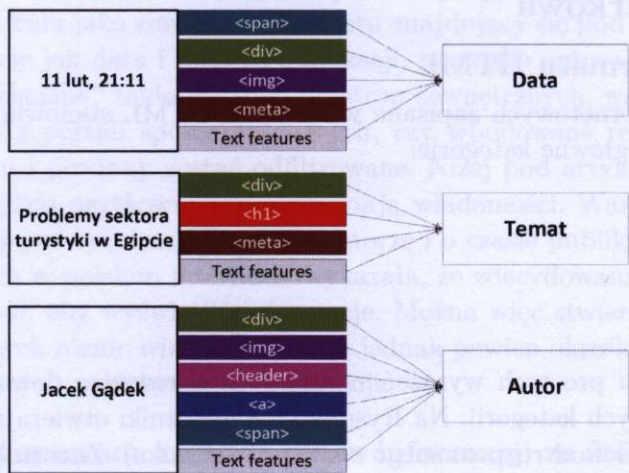
Rysunek 3.4: Wstępna sekwencja głównych kategorii w formacie HTML.
(Źródło: opracowanie własne).

Łącznie rozpoznawane są 124 znaczniki HTML. Kiedy napotkany ciąg znaków wskazuje cechy znacznika, ale nie daje się sklasyfikować do żadnego znanego rodzaju, to klasyfikowany jest jako

znacznik rodzaju nieznanego. W wyniku wstępnego sparsowania źródła HTML otrzymywana jest sekwencja tokenów, która to poddana zostaje odpowiedniemu podziałowi na zbiory tokenów, występujące w osobnych przestrzeniach międzytekstowych. Na tym etapie wstępny parser przypisuje napotkanym ciągom znaków pasującą kategorię. Tak więc w tokenie przechowywana jest jedynie informacja o wykrytej kategorii i rodzaju znacznika, reszta informacji (ciąg znaków, zawartość znacznika, itp.) jest tracona. W ten sposób unikane są niejednoznaczności i błędy występujące w formacie HTML.

3.2.2 Przestrzeń międzytekstowa w formacie HTML

Tekstem zapisanym w języku naturalnym, występujący pomiędzy znacznikami w formacie HTML nazywać będziemy przestrzenią tekstową, która oddziela kolejne przedziały przestrzeni międzytekstowej. Wystąpienie każdego przedziału przypisane jest do dyskretnej chwili czasu t , ponadto przyjmuje się, że dane zapisane w formacie HTML rozpatrywane są chronologicznie od góry do dołu; pierwszy rozpoznany zbiór występuje w chwili $t = 1$, a ostatni $t = T$. W każdym przedziale zawarte są określone znaczniki otwierające, co przedstawia Rysunek 3.5.



Rysunek 3.5: Grupowanie informacji z przestrzeni międzytekstowej.
(Źródło: opracowanie własne).

W niniejszej rozprawie przyjęto, że wystąpienie znaczników jest cechą charakterystyczną dla danego przedziału, dlatego zbiór znaczników w przestrzeni międzytekstowej może być użyty jako obserwacja (w rozumieniu obserwacji HMM). Grupowanie znaczników w danym przedziale rozpatrywana jest na dwa sposoby:

- **binarne** – na zasadzie czy dany znacznik w ogóle wystąpił w danym przedziale międzytekstowym, bez względu na kolejność czy licznosc wystąpień;
- **w wyrażenia** – znaczniki występujące w przedziale międzytekstowym tworzą unikatowe wyrażenia, z uwzględnieniem ich kolejności, jak i licznosci wystąpień.

Oba warianty poddane zostały ocenie w celu określenia, który z nich statystycznie lepiej grupuje istotne informacje dla zadania pozyskiwania informacji.

3.2.3 Mierzalne cechy przestrzeni tekstowej

Tekst w przestrzeni tekstowej otoczony znacznikami HTML posiada pewne cechy mierzalne, które dopełniają obserwację uzyskaną z przestrzeni międzytekstowej. Wybrane progi parametryzujące cechy, wykorzystywane w rozprawie w ramach prac badawczych, określone zostały w sposób empiryczny. Tabela 3.1 przedstawia zestaw cech mierzalnych dopełniających obserwację uzyskaną z przestrzeni międzytekstowej danych zapisanych w formacie HTML.

Opis cechy	Realizacja cechy
Długość tekstu w polu tekstowym	$L = \begin{cases} 2, & \text{jeżeli tekst} > 200 \text{ znaków,} \\ 1, & \text{jeżeli tekst} > 50 \text{ znaków,} \\ 0, & \text{w. p. r.} \end{cases}$
Pozycja pola tekstowego w pliku HTML	$P = \begin{cases} 2, & \text{jeżeli pozycja} > 80\% \text{ pliku;} \\ 1, & \text{jeżeli pozycja} > 20\% \text{ pliku;} \\ 0, & \text{w. p. r.} \end{cases}$
Współczynnik wystąpienia wielkich liter w polu tekstowym	$U = \begin{cases} 1, & \text{jeżeli } \frac{\text{l. wielkich liter}}{\text{l. wyrazów}} > 20\%; \\ 0, & \text{w. p. r.} \end{cases}$
Współczynnik wystąpienia znaków specjalnych (+, =, -, #, -, itp.) w polu tekstowym	$S = \begin{cases} 1, & \text{jeżeli } \frac{\text{l. znaków specjalnych}}{\text{l. wyrazów}} > 30\%; \\ 0, & \text{w. p. r.} \end{cases}$
Współczynnik wystąpienia znaków przestankowych w polu tekstowym	$I = \begin{cases} 1, & \text{jeżeli l. } \frac{\text{l. znaków przestankowych}}{\text{l. wyrazów}} > 0\%; \\ 0, & \text{w. p. r.} \end{cases}$

Tabela 3.1: Zestaw mierzalnych cech w przestrzeni tekstowej.
(Źródło: *opracowanie własne*).

W zamyśle cechy mierzalne stanowią dodatkowe informacje pozwalające lepiej zidentyfikować obserwowany obiekt. Przykładowo wysoki współczynnik wielkich liter wskazuje, że dana przestrzeń tekstowa może mieć pewne specjalne przeznaczenie jako tytuł czy temat artykułu. Z drugiej strony nadzwyczaj długi tekst w przestrzeni tekstowej połączony z krańcową pozycją w pliku HTML raczej wyklucza klasyfikację obserwacji jako tytuł.

3.3 Przygotowanie modelu HMM

W celu pozyskania treści właściwej (takiej jak temat, tekst artykułu, autorstwo itp.) z portali informacyjnych przygotowany został ukryty model Markowa. Kategorie, które utożsamiane ze stanami w ukrytym modelu Markowa przedstawia Tabela 3.2.

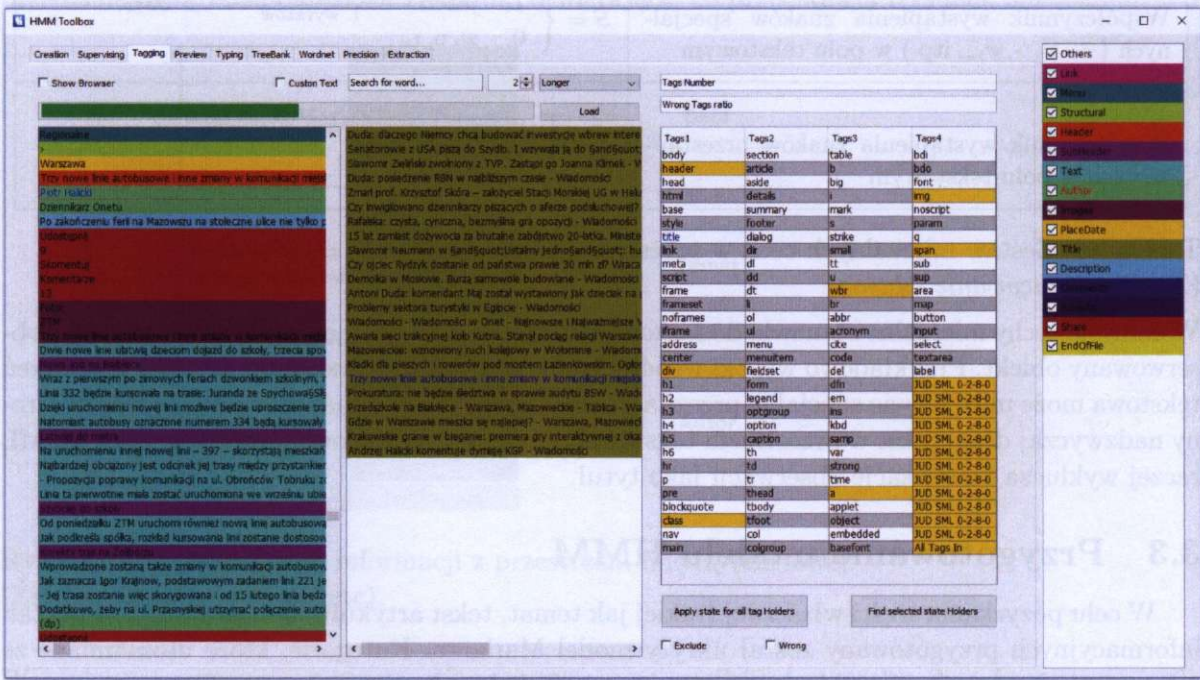
Numer	Nazwa stanu	Opis
1	Inny	elementy nieoznakowane
2	Link	hiperłącza i odnośniki do innych stron
3	Menu	elementy umożliwiające poruszanie się na stronie internetowej
4	Element szkieletowy	powtarzalne elementy tj, przyciski, etykiety, ramki, itd.
5	Temat	temat artykułu
6	Podtemat	temat rozdziału w artykule
7	Tekst właściwy	tekst właściwy artykułu
8	Autor	autor artykułu
9	Obraz	występujące obrazy i grafiki
10	Tytuł	tytuł strony internetowej
11	Miejsce i Czas	czasoprzestrzenna informacja odnośnie artykułu
12	Opis	opis treści zawartej w artykule
13	Komentarz	komentarze pozostawione przez użytkowników
14	Reklama	każda wykryta forma reklamy
15	Udostępnianie	elementy umożliwiające udostępnianie artykułu w innych serwisach
16	Koniec strony	stan ważny z punktu widzenia nauczania modelu

Tabela 3.2: Zestaw kategorii przypisanych do stanów modelu.
(Źródło: *opracowanie własne*).

Prezentowany w Tabeli 3.2 zbiór stanów określony został w sposób empiryczny na podstawie prac eksperymentalnych z wykorzystaniem różnych (mniejszych, większych) konfiguracji zbiorów.

3.4 Uczenie nadzorowane – znakowanie treści

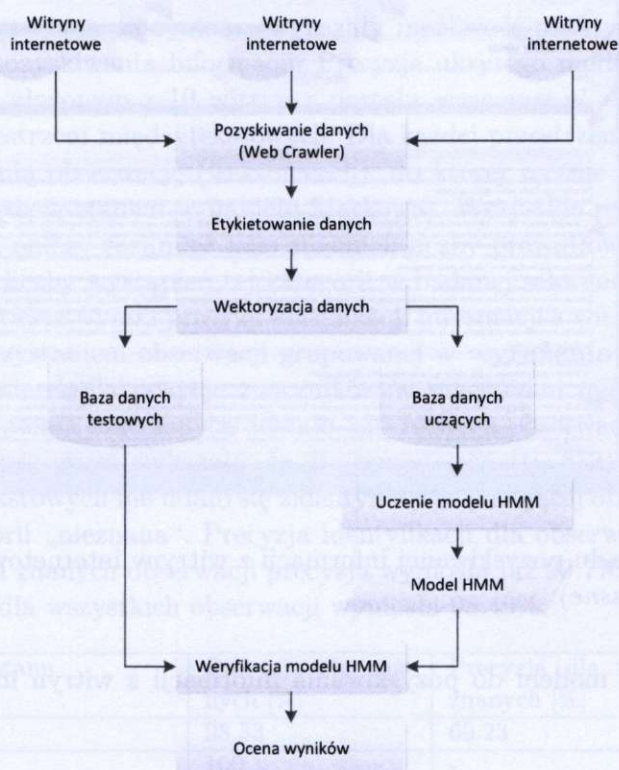
W trybie zbiorowego znakowania treści (patrz Rysunek 3.6) wyróżnić można trzy główne okna usprawniające ręczne znakowanie. Środkowe okno wyświetla zbiór wybranych przestrzeni tekstowych, dla których obserwacje spełniają zadane kryteria. Kryteria dla przestrzeni tekstowej dotyczą parametrów mierzalnych, takich jak liczba wyrazów w tekście czy też wystąpienie zadanej sekwencji liter. Okno po prawej stronie sygnalizuje wystąpienie znaczników HTML w przestrzeni międzytekstowej przypisanej do aktualnie rozpatrywanej przestrzeni tekstowej. Okno lewe zawiera kompletną sekwencję kolejnych przestrzeni tekstowych pozyskanych z pliku w formacie HTML. Okno to ukazuje otoczenie elementu wybranego z okna środkowego. Zbiorowe tagowanie odbywa się w oknie środkowym.



Rysunek 3.6: Funkcja zbiorczego oznaczania treści w HMM-Toolbox.
(Źródło: opracowanie własne).

3.4.1 Etykietowanie danych i procedura nauczania

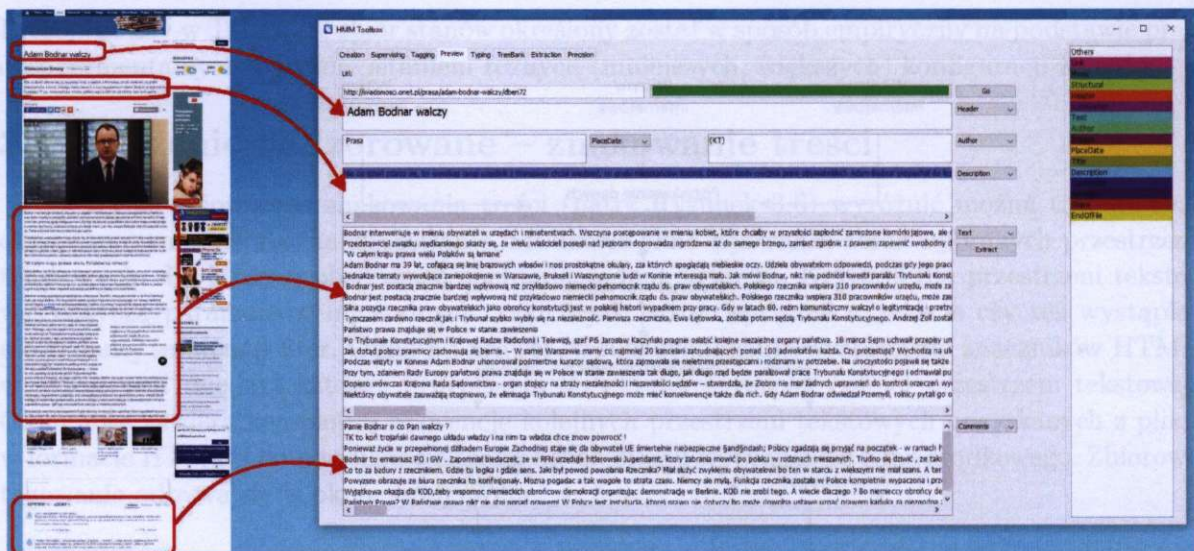
Materiał do danych uczących pozyskany został dzięki wykorzystaniu prostego robota internetowego (wbudowanego w HMM-Toolbox), który pobrał źródła stron internetowych z określonych odnośników znalezionych w żądanym portalu internetowym (procedurę pozyskiwania danych z witryn internetowych oraz nauczanie i weryfikację modelu przedstawia Rysunek 3.7). Chcąc uzyskać odpowiedni stopień uogólnienia wiedzy w modelu wymagane jest dostarczenie licznych przykładów uczących, a treści w materiale uczącym należy odpowiednio oznaczyć.



Rysunek 3.7: Procedura pozyskiwania danych z witryn internetowych oraz nauczanie i weryfikacja modelu HMM.

(Źródło: opracowanie własne).

Ręczne tagowanie wielu stron internetowych jest procesem żmudnym, więc aby usprawnić i przyspieszyć tagowanie treści procedura nauczania podzielona została na trzy fazy – wstępną, pośrednią i końcową. W ramach wstępnego uczenia baza ucząca obejmuje kilka stron z prawidłowo oznaczoną treścią. Na tej podstawie wykonywane jest pierwsze szacowanie parametrów ukrytego modelu Markowa. Ten niedoskonały model zostanie wykorzystany w fazie pośredniej do tagowania treści 10-50 stron internetowych. Na tym etapie automatycznie otagowana treść zawierać może błędy. Wszelkie niedociągnięcia należy ręcznie poprawić wykorzystując HMM-Toolbox w trybie tagowania. Posiadając kompletną bazę prawidłowo otagowanych treści dla wybranego portalu informacyjnego, przystąpić można do nauczania końcowego. W nauczaniu końcowym dokonywana jest estymacja i dopasowywanie parametrów modelu, inspirowane treningiem Viterbiego (Lember & Koloydenko 2008). W ramach weryfikacji nauczony model sprawdzany jest na grupie 10 stron referencyjnych. Sprawdzona zostaje także klasyfikacja informacji z żądanego adresu strony internetowej, wykorzystując tryb podglądu wbudowany w HMM-Toolbox, co przedstawia Rysunek 3.8.



Rysunek 3.8: Tryb podglądu pozyskiwanej informacji z witryny internetowej w HMM-Toolbox. (Źródło: *opracowanie własne*).

Procedura nauczania modelu do pozyskiwania informacji z witryn internetowych zakłada następujące etapy.

1. Faza wstępna:

- 1.1. ręczne oznakowanie 2–5 stron;
- 1.2. dokonanie pierwszego szacowania parametrów modelu.

2. Faza pośrednia:

- 2.1. automatyczne otagowanie 10-50 stron wykorzystując model z punktu 1.2;
- 2.2. ręczna poprawa i uzupełnienie niedociągnięć tagowania.

3. Faza końcowa:

- 3.1. ponowne dokonanie estymacji parametrów modelu wraz z wygładzeniem (poprawka Laplace'a);
- 3.2. zdekodowanie ścieżki stanów algorytmem Viterbiego dla treści z punktu 2.2;
- 3.3. dokonanie korekty modelu przez wypromowanie poprawnych tudzież łagodzenie błędnych stanów w ścieżce;
- 3.4. zainicjalizowanie prawdopodobieństw początkowych π wartościami na podstawie stanu „Koniec strony“.

Faza końcowa realizuje szczególny przypadek nauczania podobny do treningu Viterbiego (Lember & Koloydenko 2008). Poprawność parametrów modelu uzyskanych przy każdej ponownej estymacji zostaje zweryfikowana przez zdekodowanie ścieżki Viterbiego. Zakładając, że źródło obserwacji spełnia proces Markowa oraz że dane są prawidłowo oznaczone, należy oczekiwać, że ścieżka stanów dekodowana algorytmem Viterbiego będzie zbieżna z tą otrzymaną przy ręcznym oznaczaniu treści (bliska 100%). W stawianym zadaniu pozyskiwania informacji ze źródeł HTML, oznakowanie danych uczących pochodzących z fazy pośredniej jest ręcznie poprawiane i na tyle dobrej jakości, że przeprowadzona faza końcowa za każdym razem skutkowałą poprawą precyzji rozpoznawania treści nawet o kilka punktów procentowych.

3.5 Wyniki

Eksperymenty z prototypem systemu wykazały możliwość praktycznego zastosowania proponowanej metody pozyskiwania informacji. Precyzje ukrytego modelu Markowa sprawdzono na zbiorze testowym złożonym z 10 witryn z portalu *www.onet.pl*, dla których łącznie zidentyfikowano 5 050 przestrzeni międzytekstowych. Dla każdej przestrzeni międzytekstowej system wyznaczył odpowiednią obserwację (wektor cech), do której ręcznie przyporządkowano odpowiednią kategorię (będącą stanem w modelu Markowa). Wszystkie rezultaty liczbowe raportujące precyzje modelu należy rozumieć jako stosunek liczby prawidłowo rozpoznanych kategorii względem całkowitej liczby wystąpień tej kategorii w badanej sekwencji obserwacji.

Tabela 3.3 przedstawia wyniki precyzji klasyfikacji informacji z witryn portalu internetowego *www.onet.pl* z wykorzystaniem obserwacji grupowanej w **wyrażenia**. Tego typu rodzaj obserwacji dokładnie odzwierciedla pozycje znaczników w przestrzeni międzytekstowej w formacie HTML. W wyniku uczenia określonych zostało 530 różnych obserwacji wyrażeniowych. Ocena modelu na zestawie testowym wykazała, że dla ponad 10% (tj. 572) spośród ponad 5 tysięcy przestrzeni międzytekstowych nie udało się zidentyfikować pasującej obserwacji, stąd zostały one przypisane do kategorii „nieznana”. Precyzja identyfikacji dla obserwacji nieznanych wyniosła 65.27% natomiast dla znanych obserwacji precyzja wyniosła już 99.77%, Tym samym całkowita precyzja klasyfikacji dla wszystkich obserwacji wyniosła 98.49%.

Numer	Nazwa stanu	Precyzja dla znanych [%]	Precyzja dla nieznanych [%]	Precyzja całkowita [%]
1	Inny	98.53	69.23	97.59
2	Link	100	-	100
3	Menu	98.80	10.42	83.48
4	Element szkieletowy	100	86.11	98.40
5	Temat	100	-	100
6	Podtemat	100	-	100
7	Tekst właściwy	100	50	98.86
8	Autor	100	-	100
9	Obraz	100	-	100
10	Miejsce i czas	100	-	100
11	Tytuł	100	-	100
12	Opis	100	-	100
13	Komentarz	99.04	99.04	98.56
14	Reklama	100	-	100
15	Udostępnienie	100	0	98.92
16	Koniec strony	100	-	100
Razem		99.77	65.27	98.49

Tabela 3.3: Precyzja klasyfikacji informacji z artykułów portalu *www.onet.pl* z wykorzystaniem obserwacji grupowanej w wyrażenia.
(Źródło: *opracowanie własne*).

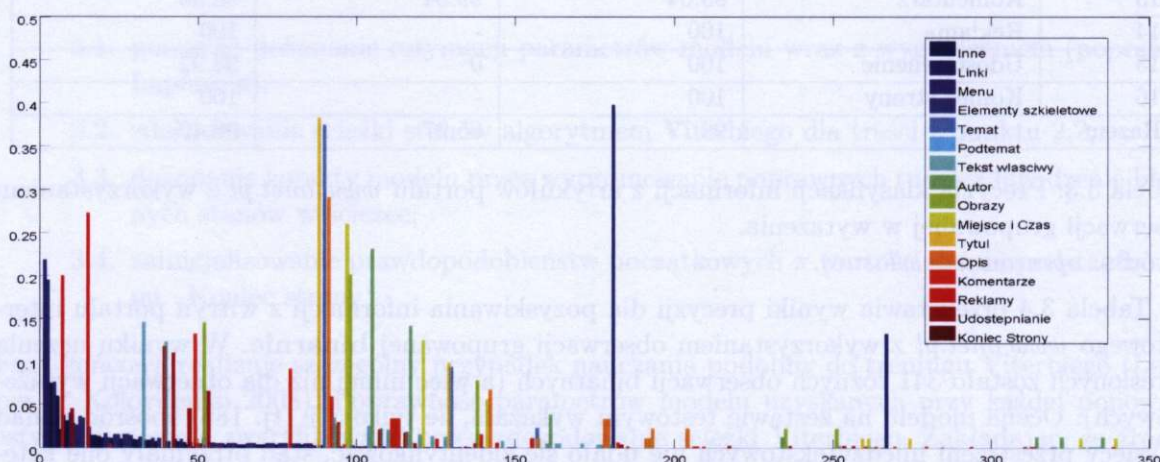
Tabela 3.4 przedstawia wyniki precyzji dla pozyskiwania informacji z witryn portalu internetowego *www.onet.pl* z wykorzystaniem obserwacji grupowanej **binarnie**. W wyniku uczenia określonych zostało 341 różnych obserwacji binarnych (a więc mniej niż dla obserwacji wyrażeniowych). Ocena modelu na zestawie testowym wykazała, że tylko 3% (tj. 185) spośród ponad 5 tysięcy przestrzeni międzytekstowych nie udało się zidentyfikować, stąd otrzymały one kategorię obserwacji „nieznana”. Precyzja identyfikacji dla obserwacji nieznanych wyniosła 74.42% natomiast dla znanych obserwacji precyzja wyniosła już 99.54%, tym samym całkowita precyzja klasyfikacji dla wszystkich obserwacji wyniosła 99.09%.

Numer	Nazwa stanu	Precyzja dla znanych [%]	Precyzja dla nieznanych [%]	Precyzja całkowita [%]
1	Inne	97.69	76.24	96.45
2	Link	100	-	100
3	Menu	98.80	35.09	95.07
4	Element szkieletowy	98.72	86.36	98.54
5	Temat	100	-	100
6	Podtemat	100	-	100
7	Tekst właściwy	98.85	100	98.86
8	Autor	100	-	100
9	Obraz	100	-	100
10	Miejsce i czas	100	-	100
11	Tytuł	100	-	100
12	Opis	100	-	100
13	Komentarz	98.97	0.00	97.96
14	Reklama	99.68	-	99.68
15	Udostępnienie	100	-	98.95
16	Koniec strony	100	-	100
Razem		99.54	74.42	99.09

Tabela 3.4: Precyzja klasyfikacji informacji z artykułów portalu *www.onet.pl* z wykorzystaniem obserwacji grupowanej binarnie.

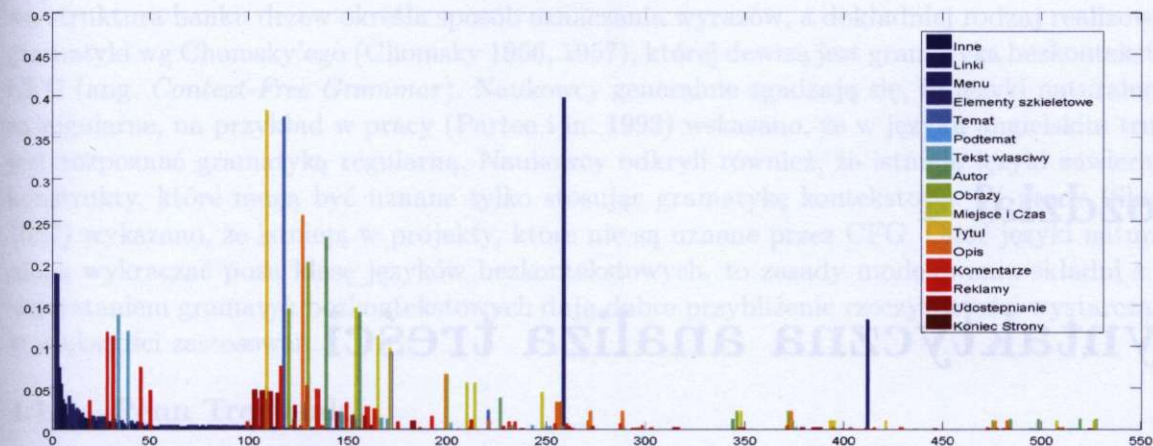
(Źródło: opracowanie własne).

W procesie nauczania modelu wektor obserwacji posortowany został począwszy od najczęściej występującej. Rysunek 3.9 i Rysunek 3.10 przedstawiają wizualizacje rozkładu prawdopodobieństwa wystąpienia obserwacji względem stanu w modelu. Kolor słupków w diagramie przypisany jest do danego stanu. Choć wizualizacje te mają charakter poglądowy, to wartym zauważenia jest fakt, że na obu diagramach pewna decydująca grupa obserwacji skupia znacząco większą masę prawdopodobieństwa od pozostałych (dominujący słupek dla danego koloru). Pomimo różnej metody grupowania znaczników w przestrzeni międzytekstowej pewne zestawy obserwacji w obu prezentowanych rozwiązaniach (tj. dla obserwacji grupowanej binarnie i w wyrażenia) wykazują podobną wrażliwość podczas zaobserwowania zadanej przestrzeni międzytekstowej.



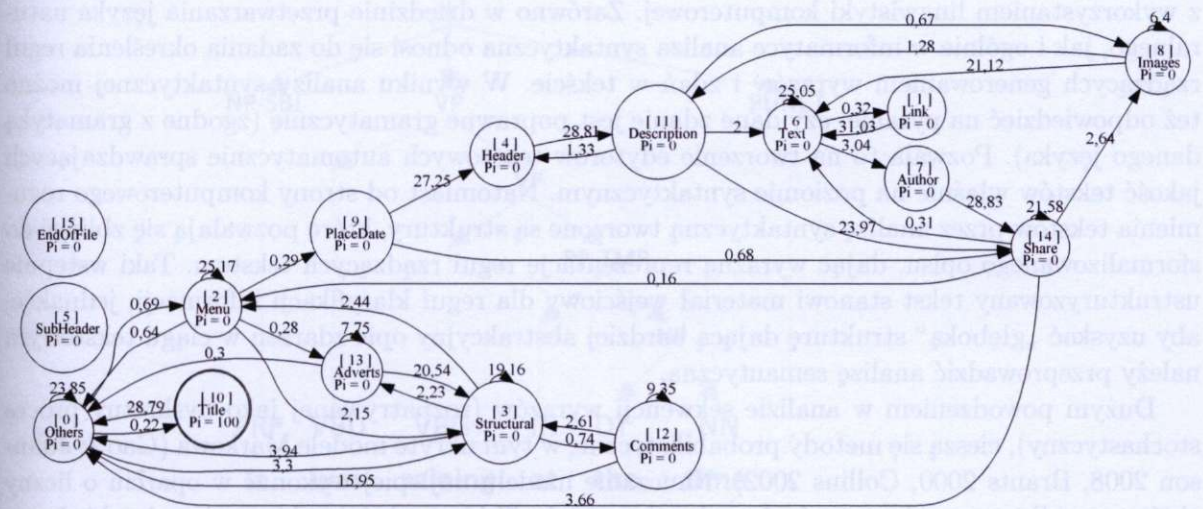
Rysunek 3.9: Wizualizacja rozkładu prawdopodobieństwa dla obserwacji grupowanej binarnie względem każdego stanu w modelu do klasyfikacji artykułów z portalu *www.onet.pl*.

(Źródło: opracowanie własne).



Rysunek 3.10: Wizualizacja rozkładu prawdopodobieństwa dla obserwacji grupowanej w wyrażenia względem każdego stanu w modelu do klasyfikacji artykułów z portalu *www.onet.pl*. (Źródło: opracowanie własne).

Zaprezentowane dwa ukryte modele Markowa nauczone zostały na tym samym zbiorze danych uczących. Różnią się one więc od siebie jedynie zestawem obserwacji (znaczniki grupowane binarnie lub grupowane w wyrażenia). Natomiast macierz prawdopodobieństw przejść jest identyczna dla obu modeli Markowa, ponieważ odtworzona została na podstawie tej samej ścieżki stanów. Rysunek 3.11 ukazuje model Markowa uzyskany dla portalu *www.onet.pl*. Widoczne na rysunku przejścia między stanami mają określony kierunek wyrażony strzałką oraz określone prawdopodobieństwo wyrażone wartością procentową. W każdym stanie widoczne jest początkowe prawdopodobieństwo wystąpienia oznaczone jako P_i . Stany pomocnicze (np. EndOfFile) oraz stany które nie wystąpiły w procesie uczenia są w modelu nieosiągalne i przedstawione jako niepołączone.



Rysunek 3.11: Model Markowa dla klasyfikacji artykułów z portalu *www.onet.pl*. (Źródło: opracowanie własne).

Wszystkie prezentowane w rozprawie grafy wizualizujące modele Markowa zostały utworzone z użyciem oprogramowania Graphviz (Ellson i in. 2001) bazując na opisie struktury (w formacie DOT) nauczonych modeli wygenerowanym przez aplikację HMM-Toolbox.

Rozdział 4

Syntaktyczna analiza treści

W tekście zapisanym w języku naturalnym wyróżnić można podstawowe znaki graficzne określające pewną funkcję fonetyczną (litery) lub strukturalną (przecinek, kropka). Sekwencja kombinacji liter tworzy wyrazy. Dla uproszczenia można przyjąć, że każdy wyraz oddzielony jest znakiem specjalnym (spacja), jednakże kombinacja wyrazów i znaków interpunkcji nie jest dowolna. Jak powszechnie wiadomo, każdy język posiada pewien zestaw reguł opisujący jego strukturę. Reguły te narzucają kategorię syntaktyczną i miejsce w szyku zdania każdemu wyrazowi. Stąd wyrazy o podobnym zachowaniu i funkcji w zdaniu są grupowane w te same kategorie (rzeczowniki, czasowniki, itd.). Do każdego wyrazu przypisana jest przynajmniej jedna kategoria syntaktyczna. Innymi słowy sekwencja liter może być interpretowana na przykład jako czasownik jak i rzeczownik w zależności od kontekstu zdania (np. „**biegli** (*rzeczownik*) sędowi zostali powołani“ vs. „chłopcy **biegli** (*czasownik*) przez pole“). Dokładne rozpoznanie kategorii syntaktycznej jest warunkiem koniecznym do określenia znaczenia wyrazu. W wyniku analizy syntaktycznej do każdego wyrazu w tekście zostanie przypisana tylko jedna kategoria syntaktyczna odsłaniająca znaczenie wyrazu użytego w zdaniu.

Analiza syntaktyczna znajduje zastosowanie w aplikacjach wyszukiwania i klasyfikacji informacji tekstowych. Do tej pory prowadzone są aktywne badania w tej dziedzinie, coraz częściej z wykorzystaniem lingwistyki komputerowej. Zarówno w dziedzinie przetwarzania języka naturalnego, jak i ogólnie w informatyce analiza syntaktyczna odnosi się do zadania określenia reguł rządzących generowaniem wyrazów i zdań w tekście. W wyniku analizy syntaktycznej można też odpowiedzieć na pytanie, czy dane zdanie jest poprawne gramatycznie (zgodne z gramatyką danego języka). Pozwala to na tworzenie edytorów tekstowych automatycznie sprawdzających jakość tekstów właśnie na poziomie syntaktycznym. Natomiast od strony komputerowego rozumienia tekstów przez analizę syntaktyczną tworzone są struktury, które pozwalają się zbliżyć do sformalizowanego opisu, dając wyraźną reprezentację reguł rządzących tekstem. Taki wstępnie ustrukturyzowany tekst stanowi materiał wejściowy dla reguł klasyfikacji informacji, jednakże, aby uzyskać „głębką“ strukturę dającą bardziej abstrakcyjny opis zdarzeń w ciągu tekstowym należy przeprowadzić analizę semantyczną.

Dużym powodzeniem w analizie sekwencji wyrazów (rozpatrywanej jako dyskretny proces stochastyczny), cieszą się metody probabilistyczne, w tym ukryte modele Markowa (Gao & Johnson 2008, Brants 2000, Collins 2002). Nauczanie modelu najlepiej wykonać w oparciu o liczny zbiór prawidłowo oznakowanych danych tekstowych. Takimi właśnie zbiorami są banki drzew (ang. *treebank*), które opisują strukturę zdania i części mowy.

W niniejszej rozprawie analiza syntaktyczna przeprowadzona jest zarówno dla języka angielskiego, jak i polskiego, stąd prezentowane są tu dwa banki drzew: Penn Treebank dla języka angielskiego oraz Składnica frazowa dla języka polskiego.

4.1 Bank drzew składniowych

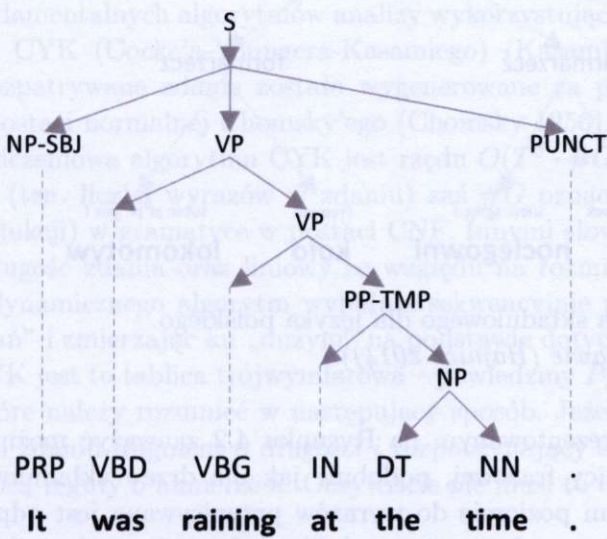
Bank drzew jako sparsowany i oznakowany korpus danych tekstowych stanowi doskonały materiał do nauczania ukrytych modeli Markowa, jak również ich późniejszej weryfikacji. Frazo-

wa struktura banku drzew określa sposób oznaczania wyrazów, a dokładniej rodzaj realizowanej gramatyki wg Chomsky'ego (Chomsky 1956, 1957), której dewizą jest gramatyka bezkontekstowa CFG (ang. *Context-Free Grammar*). Naukowcy generalnie zgadzają się, że języki naturalne nie są regularne, na przykład w pracy (Partee i in. 1993) wskazano, że w języku angielskim trudno jest rozpoznać gramatykę regularną. Naukowcy odkryli również, że istnieją języki zawierające konstrukty, które mogą być uznane tylko stosując gramatykę kontekstową. W pracy (Shieber 1987) wykazano, że istnieją w projekty, które nie są uznane przez CFG. Choć języki naturalne mogą wykraczać poza klasę języków bezkontekstowych, to zasady modelowania składni z wykorzystaniem gramatyk bezkontekstowych dają dobre przybliżenie rzeczywistości wystarczające w większości zastosowań.

4.1.1 Penn Treebank

W ramach projektu OntoNotes Release 5.0 (Ralph Weischedel 2013) dokonano oznaczenia i integracji różnorodnych zbiorów tematycznych zapisanych w głównych językach (angielski, chiński i arabski). W skład korpusu wchodzi między innymi bank drzew Penn Treebank. Baza ta zawiera łącznie około 800 tys. wyrazów w języku angielskim, zapisanych w formie drzew składniowych. Dla przypomnienia struktury drzewa składniowego z zasobów Penn Treebank rozpatrzmy zdanie:

„It was raining at the time.“
 (TOP (S (NP-SBJ (PRP It))
 (VP (VBD was)
 (VP (VBG raining)
 (PP-TMP (IN at)
 (NP (DT the)
 (NN time))))))
 (. .)))



Rysunek 4.1: Przykład drzewa składniowego dla języka angielskiego.
 (Źródło: opracowane na podstawie (Taylor i in. 2003)).

Rozkład zdania wizualizuje Rysunek 4.1. Łatwo zauważyć, że zdanie reprezentowane jest w sposób hierarchiczny. Taka struktura hierarchiczna nazywana jest drzewem. Na najniższym poziomie określana jest część mowy przypisana bezpośrednio do danego wyrazu, a na poziomie wyższym syntaktyczne znaczenie występujących fraz. Na początkowym etapie badań w dziedzinie lingwistyki duży nacisk kładziono na składniki gramatyczne. Założono, że zapisy w języku

naturalnym można przedstawić na zasadzie hierarchii pogrupowanych komponentów składniowych. Grupy te mogą się częściowo pokrywać oraz zawierać kolejne zagnieżdżone grupy, aż do elementów atomowych – wyrazów zdania.

4.1.2 Składnica frazowa

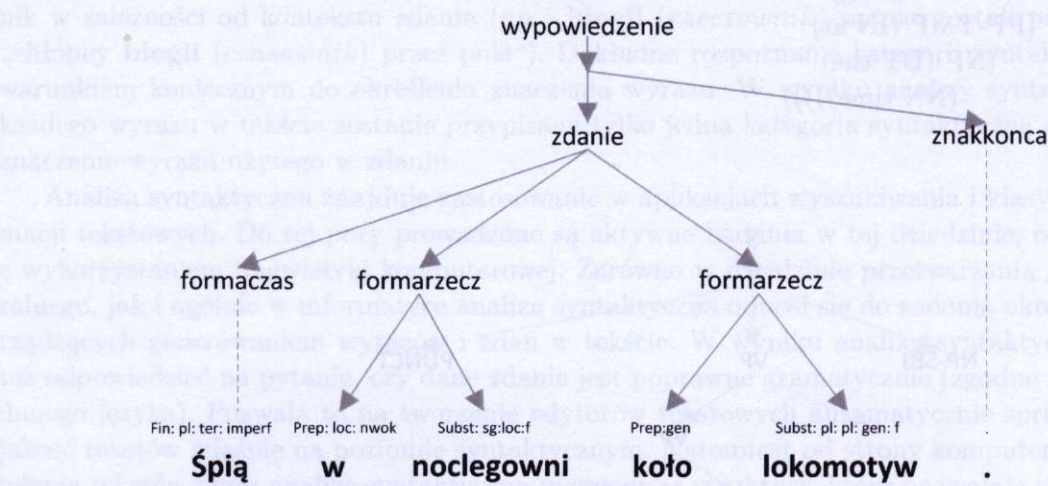
W ramach projektu N-N104-224735 utworzony został bank drzew składniowych dla języka polskiego z wykorzystaniem automatycznej analizy składniowej. Do określenia składni zdania zastosowano parser składniowy Świgr z wbudowanym słownikiem walencyjnym (Woliński 2004). Dodatkowo wygenerowane drzewa składniowe zostały przeanalizowane i zatwierdzone przez zespół specjalistów zwanych lingwistycznymi dendrologami. Baza ta zawiera ponad 200 tys. wyrazów w języku polskim zapisanych w formie drzew składniowych.

Dla przypomnienia struktury drzewa składniowego z zasobów Składnicy frazowej rozpatrzmy zdanie:

„Śpią w noclegowni koło lokomotyw.”

trees/NKJP_1M_1303900001/morph_41-p/morph_41.12-s.xml.tree

1	Śpią	spać	fin	fin	pl ter imperf	0	-	-	-
2	w	w	prep	prep	loc nwok	1	-	-	-
3	noclegowni	noclegownia	subst	subst	sg loc f	2	-	-	-
4	koło	koło	prep	prep	gen	1	-	-	-
5	lokomotyw	lokomotywa	subst	subst	pl gen f	4	-	-	-
6	.	.	punct	punct		0	-	-	-



Rysunek 4.2: Przykład drzewa składniowego dla języka polskiego. (Źródło: opracowane na podstawie (Hajnicz 2014)).

W rozkładzie zdania zaprezentowanym na Rysunku 4.2 zauważyć można również hierarchiczną strukturę. W Składnicy frazowej, podobnie jak dla drzew składniowych w zasobach Penn Treebank, na najniższym poziomie do wyrazów przypisywana jest odpowiadająca część mowy wyróżniając dodatkowo szeroką grupę modyfikatorów m.in. rodzajnik, przypadek, czas itp. Na poziomie wyższym określone są frazy (np. formaczas – fraza odczasownikowa, formarzecz – fraza odrzeczownikowa) biorące udział w formowaniu zdania.

4.1.3 Metody konstruowania drzew składniowych

Ta sekcja opisuje problemy napotymane podczas analizowania tekstu w języku naturalnym, z powodu których klasyczne algorytmy analizy dla gramatyk bezkontekstowych nie są stosowane, oraz krótko opisuje efektywne algorytmy konstruowania drzew składniowych. Zakładając

odpowiedni stopień przybliżenia i stosując odpowiednie algorytmy rzutowania, gramatyka bezkontekstowa może być wykorzystana zarówno dla języków naturalnych, jak i formalnych. Jednakże dla formalnych języków – takich jak języki programowania – zazwyczaj analiza możliwa jest do efektywnego przeprowadzenia w czasie liniowym zależnym od długości łańcucha wejściowego. Analiza języków naturalnych ze względu na ich wysoki stopień niejasności jest bardziej wymagająca (Jurafsky & Martin 2000).

Dokonując rozbioru morfologicznego analizowanych wyrazów, możliwe jest rozróżnienie cech morfologicznych (leksykalnych i fleksyjnych). Automaty przetwarzania gramatyki bezkontekstowej zazwyczaj zawierają dużą liczbę niedeterministycznych reguł, co w połączeniu z klasycznymi algorytmami rzutowania prowadzi do licznych zapętleń, a w konsekwencji niskiej sprawności analizy. Dodatkowo gramatyka języków naturalnych zazwyczaj dopuszcza kilka możliwych reprezentacji składni. Należy pamiętać, że istnieją przypadki zdań, które są poprawne pod względem syntaktyki, a mimo to semantycznie błędne. Prosty przykładem jest zdanie: „*Twarde niebo spadło na mleko w dominującej poszewce.*”. Choć syntaktycznie zdanie wydaje się prawidłowo skonstruowane, to jego abstrakcyjna treść jest pozbawiona sensu i nie wnosi żadnej pragmatycznej informacji.

Przez odpowiednią analizę zależności językowych i znajomości semantyki słów dokonać można ujednoznacznienia homonimów. Natomiast odkrycie znaczenia homonimów często wymaga znajomości pragmatyki tekstu lub jego pozajęzykowego kontekstu (Ciecierski 2011). W przypadku braku wiedzy na temat semantyki i pragmatyki parser syntaktyczny w oparciu o wbudowane morfologiczne reguły lub struktury składniowe zdania albo przedstawi wszelkie możliwe rozwiązania, lub też jedno najlepsze według jakiegoś kryterium.

Ze względu na problemy, które pojawiają się w trakcie analizy tekstów metodą gramatyk bezkontekstowych (duża liczba zwrotów i konieczność przechowywania wielu opcji analizy), naukowcy zaproponowali szereg algorytmów, które używają zasad programowania dynamicznego. Główną ideą programowania dynamicznego w analizie za pomocą gramatyk bezkontekstowych jest kompaktowe przechowywanie wcześniej przetwarzanych poddrzew tak, że w przypadku nawrotu nie ma potrzeby ponownego wykonywania obliczeń dla odwiedzonej gałęzi poddrzewa.

Jednym z fundamentalnych algorytmów analizy wykorzystującym programowanie dynamiczne jest algorytm CYK (Cocke'a-Youngera-Kasamiego) (Kasami 1966, Younger 1967), który sprawdza, czy rozpatrywane zdanie zostało wygenerowane za pomocą danej gramatyki bezkontekstowej w postaci normalnej Chomsky'ego (Chomsky 1956).

Złożoność obliczeniowa algorytmu CYK jest rzędu $O(T^3 \cdot \#G)$, gdzie T jest długością parsowanego zdania (tzn. liczbą wyrazów w zdaniu) zaś $\#G$ oznacza rozmiar gramatyki – czyli liczbę reguł (produkcji) w gramatyce w postaci CNF. Innymi słowy jest to algorytm sześcienny ze względu na długość zdania oraz liniowy ze względu na rozmiar gramatyki. Zgodnie z ideą programowania dynamicznego algorytm wypełnia sekwencyjnie pewną tablicę, poczynając od „małych rozwiązań” i zmierzając ku „dużym” na podstawie dotychczasowych wpisów w tablicy. W algorytmie CYK jest to tablica trójwymiarowa – powiedzmy $P_{T \times T \times \#G}$ – przechowująca wartości logiczne, które należy rozumieć w następujący sposób. Jeżeli $P[i, j, k] = \text{true}$, to oznacza to, że w badanym zdaniu fragment o długości i rozpoczynający się od pozycji j można wyprodukować za pomocą reguły o numerze k . Oczywiście nie musi to być jedyna reguła pozwalająca na tę produkcję.

Inicjalizacja algorytmu polega na wypełnieniu całej tablicy wartościami false, a następnie zbadaniu możliwych produkcji jednoelementowych (terminalnych) na kolejnych pozycjach – czyli ustawieniu $P[1, j, k] = \text{true}$, wszędzie tam gdzie dla wyrazu (symbolu terminalnego) stojącego na pozycji j istnieje reguła (nr k), która go produkuje. Główna część algorytmu to poczwórnie zagnieżdżona pętla przebiegająca kolejno: coraz większe długości fragmentów zdania ($i = 2, \dots, T$), możliwe pozycje startowe ($j = 1, \dots, T - i + 1$) dla tych fragmentów, możliwe podziały ($p = 1, \dots, i - 1$) aktualnej długości fragmentu na dwie niepuste części odpowiednio o długościach p oraz $i - p$, i wreszcie zbiór reguł nieterminalnych – powiedzmy postaci $R_k \rightarrow R_q R_r$. Taki zestaw pętli wyjaśnia wspomnianą wcześniej złożoność obliczeniową. Krok

indukcyjny (wewnątrz pętli) pozwala ustawić wartość true dla komórki $P[i, j, k]$ wtedy, i tylko wtedy, gdy $P[p, j, q] = \text{true}$ oraz $P[i - p, j + p, r] = \text{true}$. Innymi słowy ma to miejsce, gdy wiemy, że fragment rozpięty pomiędzy indeksami j oraz $j + i - 1$ pozwala się zbudować z dwóch części: lewej (rozpiętej pomiędzy j oraz $j + p - 1$) za pomocą reguły R_q oraz prawej (rozpiętej pomiędzy $j + p$ oraz $j + i - 1$) za pomocą reguły R_r . Ostateczna odpowiedź algorytmu jest twierdząca – badane zdanie można wyprodukować za pomocą danej gramatyki G – jeżeli $P[T, 1, 1] = \text{true}$ (gdzie trzeci indeks równy 1 jest związany z regułą uznaną za startową).

S	→ subj	comp	S					
pred	→ verb	comp						
comp	→ prep	adjunct						
pred	→ subj	verb						
subj	→ subst	adjunct	pred					
adjunct	→ adj	subst						
adjunct	→ adv	verb						
			subj		comp			
subst	→ Ignacy				adjunct			
adv	→ chętnie						adjunct	
verb	→ opowiada							
prep	→ o		subst		adv		verb	
adj	→ swojej						prep	
subst	→ rodzinie		Ignacy		chetnie		opowiada	
							o	
							swojej	
							rodzinie	

Rysunek 4.3: Tabela odkrywania wyrażeń nieterminalnych algorytmem CYK.

(Źródło: opracowanie własne).

Prosty przykład działania algorytmu CYK zobrazowano na Rysunku 4.3. Lewa strona rysunku przedstawia pewną gramatykę, zaś prawa wynikową zawartość tablicy wypełnionej w wyniku działania algorytmu CYK dla zdania „Ignacy chętnie opowiada o swojej rodzinie“. Algorytm wypełnia tablicę idąc wierszami od dołu do góry, i w ramach każdego wiersza od lewej do prawej. Obecność pewnej reguły (produkcji) w komórce tablicy symbolizuje przypisanie do niej wartości logicznej true.

Algorytm, który również wykorzystuje zasady programowania dynamicznego zaproponowany przez Earleya (Earley 1970) wykonuje analizę „z góry na dół”. Używa on tablicowych metod analizy składniowej do przechowywania stanu pośredniego drzewa. Z określonej pozycji algorytm na podstawie zstępujących przewidywań skanuje wyrażenia. Analiza rozpoczyna się od skanowania tzw. sytuacji Earleya, a następnie algorytm na podstawie bieżącego stanu i bieżącego wyrazu wykonuje etap przewidywania kolejnych produkcji gramatyki. Analiza zostanie ukończona, jeżeli ostatni rekord tabeli wskazuje symbol startowy gramatyki. Algorytmy CYK, Earleya i inne odmiany tablicowych metod analizy w najgorszym wypadku mają złożoność $O(T^3)$, gdzie T to liczba wyrazów w zdaniu.

Zasadniczo odmienne podejście do analizowania niejednoznacznych gramatyk zaproponował M. Tomita w analizatorze GLR (ang. *Generalized Left-to-Right most derivation parser*) (Tomita 1984). Analizator działa na podobnych zasadach jak analizatory LR (z lewej do prawej strony), które są przeznaczone do analizowania gramatyk. Niemniej jednak analizator GLR, aby skutecznie obsługiwać niejasności gramatyki, zapewnia możliwość rozgałęzienia stosu, gdzie przechowywana jest lista odwiedzonych stanów. Jak uprzednio wspomniano – w innych analizatorach obecność rozgałęzień zgodnie z zasadami programowania dynamicznego nie prowadzi do powielania informacji na stosie, ze względu na jego reprezentację w postaci drzewa.

Dla automatu niedeterministycznego w najgorszym przypadku złożoność algorytmów GLR jest również $O(T^3)$, natomiast w przypadku automatu deterministycznego, złożoność algorytmu wynosi $O(T)$.

Tworzenie gramatyk bezkontekstowych w celu analizowania języka naturalnego jest, jak wi-
dać, bardzo czasochłonnym zadaniem. W rzeczywistej aplikacji gramatyka musiałaby zawierać
tysiące reguł, co jest dość trudne z punktu widzenia organizacji i utrzymania. Kolejną fundamen-

talną wadą gramatyk bezkontekstowych jest fakt, że wiele rodzajów rozgałęzień drzewa może być prawdą z gramatycznego punktu widzenia, ale nie są dozwolone pod względem semantyki. Wybór odpowiedniej opcji analizy dokonywany jest na podstawie kryteriów w postaci chociażby rangi uzyskiwanej w procesie rozgałęziania. Do określenia wartości rankingu powszechnie przyjęte jest podejście statystyczne. Najbardziej znanym statystycznym podejściem do określania rangi elementów drzew jest zastosowanie probabilistycznych bezkontekstowych gramatyk PCFG (ang. *Probabilistic Context-Free Grammar*) (Collins 2003). Gramatyka PCFG różni się od zwykłych gramatyk bezkontekstowych tym, że do każdej reguły gramatyki przypisane jest prawdopodobieństwo. Zadaniem parsera w tym przypadku jest znalezienie najbardziej prawdopodobnego wariantu składni. Pierwsza gramatyka PCFG opierała się głównie na składniowych regułach i informacji na temat zgodności części mowy, jednak eksperymenty wykazały, że informacje statystyczne nie wystarczają, aby rozwiązać niejednoznaczność języka naturalnego. Oprócz reguł składniowych i zgodności części mowy należy jeszcze uwzględnić statystyki wystąpienia wyrazu. Modyfikacja ta znacznie poprawiła jakość probabilistycznych metod analizowania.

Podejścia proponowane przez naukowców różnią się techniką określania prawdopodobieństwa drzew składniowych, a także sposobami rozwiązywania problemów poszukiwania drzewa z prawdopodobieństwem maksymalnym. Z jednej strony pozwala to wybrać najlepszy rezultat według pewnego kryterium, a z drugiej strony może przyspieszyć prace analizatora określając heurystycznie najbardziej prawdopodobne drzewa składni.

4.2 Bank drzew zależności

Większość modeli struktur zależnościowych oparta jest na gramatyce zależności, dla której fundamentalne są prace L. Tesnière'a (Tesnière 1969). Gramatyka zależności sugeruje, że treść zdania może być przedstawiona w formie drzewa zależności, w którym słowa są powiązane kierunkowo, odsłaniając swoje podporządkowanie (Mielczuk 1988, Karlsson 1990). Rodzaje połączeń mogą określać funkcje gramatyczne wyrazów w zdaniu lub ogólne semantyczne relacje między wyrazami. Istniejące teorie gramatyczne niejednoznacznie określają zasady połączeń i rodzaje stosunków składniowych. Istnieją jednak pewne ogólne zasady regulujące zależności składniowe między wyrazami takie jak:

- **związek zgody**, gdzie kategoria syntaktyczna dla wyrażenia określającego jest w zgodzie z wyrazem określanym,
- **związek rzędu**, gdzie kategoria syntaktyczna dla wyrażenia określającego jest narzucona przez wyraz określany,
- **związek przynależności**, gdzie kategoria syntaktyczna dla wyrażenia określającego nie ulega zmianie (jest nieodmienna) i tylko biernie przynależy do wyrazu określanego (Bąk 2004).

Wybór drzewa zależności lub drzewa składniowe do opisu struktury składniowej zdań związany jest z właściwościami projekcyjnymi języka, jego składni, oraz dostępności szyku wyrazów. Gramatyka frazowa jest powszechnie stosowana do analizy języków z ustalonym szykiem wyrazów, których projekcja może naruszać własności gramatyki, ponieważ składniowe relacje między słowami zazwyczaj nie poddają się projekcji. Przykładem takich grup są języki angielski czy niemiecki. Uważa się, że gramatyki zależności dobrze odzwierciedlają specyfikę języków o swobodnym szyku wyrazów, w których znaczna ilość połączeń nieprojekcyjnych może występować między słowami. Języki te to przykładowo czeski, polski, ukraiński i inne języki wschodniosłowiańskie.

Warto podkreślić, że obecność swobodnego szyku wyrazów w języku naturalnym wpływa na złożoność jego opisu przy użyciu różnych formalizmów. W celu analizy struktury w językach ze swobodnym szykiem wyrazów gramatyka frazowa może wymagać dużej liczby reguł, tak aby pokryć każdą możliwą kombinację wyrazów (Jurafsky & Martin 2000). Natomiast reguły gramatyki zależnościowej najczęściej są niezależne od szyku wyrazów.

4.2.1 Metody konstrukcji drzew zależności

Pierwsze drzewa zależności, jak również sposoby konstruowania składników drzew były oparte na ręcznie ustalonym formalizmie gramatyki. Istnieje kilka klas formalizmów gramatyki zależności zadań, jak i podejść do budowania drzewa zależności. Pierwsza klasa obejmuje formalizmy, które są bliskie gramatyce formalnej Chomsky'ego (Hays 1964, Gaifman 1965) i są w stanie modelować projekcyjne struktury językowe. Syntaktyczna analiza dla tego typu podejścia wykonywana jest, aby rozwiązać problem dwuznaczności i nieokreśloności, często w jednakowy sposób, stosując algorytmy programowania dynamicznego do budowy komponentów drzewa, jak algorytm Earley'a lub CYK (Lombardo & Lesmo 1996). Inna klasa formalizmów zakłada gramatykę zależności jako zbiór ograniczeń, który pozwala modelować nieprojekcyjne struktury językowe. Analiza z punktu widzenia formalizmu polega na wyczerpującym przeszukaniu drzewa zależności tak, aby zaspokoić wszystkie ograniczenia gramatyki.

Inne podejście do reprezentacji gramatyki zależności opracowane przez M. Covingtona (Covington 2001) polega na ustawieniu, niektórych zasad systemu (zestaw funkcji logicznych) i algorytmu deterministycznego do analizowania tekstu. Reguły i algorytmy jednoznacznie definiują składnię drzewa zależności. Prosty algorytm parsowania o złożoności $O(T^2)$: począwszy od pierwszego słowa w zdaniu, skanuje słowa od lewej do prawej i ustala relację między słowem bieżącym i pozostałymi słowami. Covington wykazał, że ten algorytm i zestaw funkcji logicznych pozwalają budować poprawne składniowo drzewa bardziej złożonych formalizmów gramatyki zależności.

Do tej pory przedmiotem badań lingwistyki komputerowej jest poszukiwanie najlepszej formalizacji jako gramatyki zależności. Zastosowanie maszynowego uczenia z nauczycielem w celu budowania drzew zależności wykazało szereg zalet w porównaniu do rozwiązań opartych na ręcznie skompilowanej gramatyce, a wśród nich – redukcję złożoności tworzenia analizatora dla obszernego korpusu znaczników. Podstawowe podejście do budowania drzewa zależności, które używa technik uczenia maszynowego, wykorzystuje wykres zależności oparty na wystąpieniach i przejściach.

4.2.2 Podejście oparte o wykres zależności

W podejściu opartym o wykres zależności przyjęto, że model jest tworzony na podstawie ilościowej informacji z drzewa składni (wykresy z zależnościami). Zazwyczaj w modelu zależności wyróżnić można elementarne połączenia (w modelach pierwszego rzędu – jeden łuk, drugiego – para sąsiednich łuków), do których przypisane są wagi. Analizowanie problemu w tym przypadku sprowadza się do wyczerpującego przeszukania wykresu drzewa zależności i oszacowania zależności o najwyższej wadze. Jedną z pierwszych prac, która opisała podejście do analizowania wykresu było dzieło J. Eisnera (Eisner 1996). Do oceny wykresu zależności używane są modele generatywne, gdzie pod uwagę brany jest iloczyn prawdopodobieństw wystąpienia słowa, wagi łuków, jak i oceny etykiet składniowych łuków i ocena całego drzewa. Eisner zaproponował model generatywny do oceny wykresu zależności i algorytm parsowania podobny do algorytmu CYK. Pozwoliło to na tworzenie projekcyjnych drzew składniowych zależności w czasie $O(T^3)$.

W (McDonald i in. 2005) zastosowano analizator znany w literaturze jako parser MST (ang. *Maximum Spanning Tree*), maksymalnie rozpinający algorytm parsowania. Stwierdzono że złożoność obliczeniowa budowy dowolnego (nieprojekcyjnego) drzewa z zastosowaniem parsera MST wynosi $O(T^2)$.

Badacze wykazali także, że ta modyfikacja pozwala na osiągnięcie lepszej jakości parsowania dla języków ze swobodnym szykiem wyrazów. Wydajność algorytmu MST jest tu wyższa w porównaniu z podejściem, w którym stosuje się algorytm Eisnera. Jednak użycie algorytmu wyszukiwania analizatorem MST może prowadzić do nieznacznego spadku jakości analizy języków, w których liczba połączeń nieprojekcyjnych jest mała (na przykład angielski).

W badaniach (McDonald & Satta 2007) z wykorzystaniem modelu drugiego rzędu do oceny wykresu zależności wykazano, że dla nieprojekcyjnych struktur drzew wyszukiwanie zależności jest problemem NP-pełnym. Dlatego też badania w tym obszarze w większości ograniczały

się do projekcyjnych struktur językowych. Algorytm przetwarzania zaproponowany przez McDonald i Pereira (Nakagawa 2007) pozwalał na budowę drzewa w zależności w czasie $O(T^3)$, natomiast w pracy (Carreras 2007) uzyskał złożoność obliczeniową wyniosła $O(T^3)$ dla modeli drugiego rzędu.

4.2.3 Podejście oparte o model przejść

Na ogół algorytm oparty na modelu przejść jest zredukowany do prognozowania przejścia ze stanu bieżącego do nowego stanu. Procedura przewidywania i przejścia występuje do czasu, aż kompletna składnia drzewa zależności zostanie zbudowana. Podejście to zostało pierwotnie zaproponowane w pracy (Kudo & Matsumoto 2000), gdzie użyto algorytmów takich jak „shift-reduce“ o złożoności $O(T^2)$, pozwalających na określenie składni projekcyjnych drzew zależności. Natomiast do przewidywania rozgałęzień analizatora zastosowano maszynę wektorów nośnych SVM (ang. *Support Vector Machine*). Podejście oparte na modelu przejść zostało opracowane przez J. Nivre’a (Nivre 2005a,b, Nivre & Nilsson 2005). Zaproponował on algorytm konstruowania projekcyjnych drzew składni w czasie $O(T)$. W przeciwieństwie do (Yamada & Matsumoto 2003) analizator Nivre’a pozwala nie tylko na zbudowanie drzewa składniowego, ale także określa rodzaj powiązań i przypisuje je do etykiet. To pozwoliło na uwzględnienie w modelu prognozowania możliwości przejść, które mogą być wydobyte z częściowo skonstruowanego drzewa składni. Początkowo analizator do prognozowania przejścia korzystał z parametrycznych metod klasyfikacji, jednakże maszyna wektorów nośnych dawała najlepsze rezultaty. Algorytm Nivre’a przetwarza słowa w zdaniu kolejno z lewej do prawej, wykorzystuje stos i zapewnia następujące rodzaje przejść:

- **przesuń lewy** – ustalenie relacji między słowem wejściowym (głównym) i słowem na szczycie stosu (podrzednym), zdjęcie jednego słowa ze stosu;
- **przesuń prawy** – ustalenie związku między słowem w górnej części stosu (głównym) i słowem wejściowym (podrzednym), umieszczenie słowa wejściowego na stosie;
- **zmniejszenie** – zepchnięcie jednego słowa ze stosu;
- **zwiększenie** – umieszczenie słowa na stosie.

Analizatory w kroku „przesuń-prawy“ i „przesuń-lewy“ również odwołują się do klasyfikatora, aby określić typ składni w kontekście. Algorytm gwarantuje budowę projekcyjnej struktury drzewa. Podejście do parsowania oparte na przejściach zostało zaimplementowane w pakiecie oprogramowania MaltParser (Nivre i in. 2007). Pakiet ten obejmuje różne algorytmy, w tym algorytm Nivre’a i Covingtona, jak również realizację kilku metod uczenia maszynowego dla klasyfikatora predykcji. Z powodu jego skuteczności i wydajności MaltParser do dziś jest jednym z najczęściej stosowanych analizatorów. Prace badawczo-rozwojowe wykorzystujące aplikację MaltParser realizowane są dla wielu języków, w tym polskiego (Wróblewska & Woliński 2012).

4.3 Składnica – typy zależnościowe w języku polskim

Składnica zależnościowa zawiera bank drzew dla języka polskiego opisujący relacje między wyrazami. Zgodnie z podziałem przedstawionym w pracy (Wróblewska 2012) typy relacji pogrupowane są na argumentowe, nieargumentowe i koordynujące i odzwierciedlają funkcje, jaką dany związek wyrazowy pełni w zdaniu. Ponieważ w rozprawie nauczanie ukrytych modeli Markowa dla języka polskiego oparte jest właśnie o dane zawarte w banku drzew, to i nazewnictwo kategorii zależnościowych (reprezentowanych przez stany w modelu) jest zbieżne z nomenklaturą stosowaną w Składnicy. Stąd, typy relacji określane są przez anglojęzyczne akronimy, które nie zawsze w pełni oddają znaczenie kategorii gramatycznej. W celu klarownego przedstawienia znaczenia rozpoznawanych typów relacji zależnościowych, każdy z akronimów został wypunktowany i szczegółowo opisany z uwzględnieniem przykładowych reprezentacji.

4.3.1 Typy argumentowe

comp (dopełnieniu dalsze wyrażone częścią mowy inną niż rzeczownik)

- przymiotnikowe dopełnienie regulowane formą czasownika,
- przysłówkowe dopełnienie regulowane formą czasownika,
- nominalne dopełnienie regulowane formą czasownika, przymiotnikiem, przyimkiem lub liczebnikiem,
- przyimkowe dopełnienie regulowane formą czasownika, przymiotnikiem, przysłówkiem.

comp_fin (dopełnienie zdaniowe)

- realizowane przez związek frazeologiczny (deklaratywny, pytający lub wykrzyknikowy) uwzględniający podmiot,
- dopełnienie podrzędne regulowane formą czasownika, nadrzędnym połączeniem lub rzeczownikiem.

comp_inf (dopełnienie bezokolicznikowe)

- realizowany przez bezokolicznikowy związek frazeologiczny, regulowany:
 - frazą odrzeczownikową,
 - formą czasownika,
 - frazą odprzymiotnikową,

obj (dopełnienie bliższe)

- realizowane przez frazę odrzeczownikową odmienioną w bierniku, dopełniaczu, lub celowniku,
- regulowane formą czasownika,
- przekształcane na podmiot w konstrukcjach pasywnych.

obj_th (dopełnienie dalsze wyrażone rzeczownikowo)

- realizowane przez frazę odrzeczownikową odmienioną w celowniku,
- regulowane formą czasownika,
- spełniające rolę semantyczną: odbiorcy, doświadczającego, beneficjenta etc.,
- niepodlegające przekształceniu do podmiotu w konstrukcjach pasywnych.

pd (orzecznik, predykatyw lub wyrażenie predykatywne)

- realizowany przez każdy element w pozycji orzecznika w zdaniu,
- regulowany formą czasownika „być“.

subj (podmiot)

- część zdania oznaczająca wykonawcę czynności wyrażonej orzeczeniem,
- część zdania realizowana przez frazę odrzeczownikową, odprzymiotnikową, odliczebnikową.

4.3.2 Typy nieargumentowe

adjunct (przydawka albo okolicznik)

- odpowiada na pytania: jaki? jaka? jakie? który? która? które? czyj? czyja? czyje? ile? ilu? czego? z czego?
- realizowana jako:
 - przymiotnik zależny od rzeczownika lub cyfry,
 - przysłówki zależny od formy czasownika lub innego przysłówka,
 - fraza przymiotnikowa lub przyimkowa.

app (apozycja)

- realizowane jako fraza rzeczownikowa zależna od bezpośrednio poprzedzającego rzeczownika o tym samym znaczeniu. np. Prezydent Andrzej Duda.

complm (partykuła, spójnik podrzędny)

- realizowany przez partykułę np. że, iż, żeby, aby, by, żem,
- wprowadza klauzulę dopełniacza, poprzedzoną orzeczeniem.

mwe (wyrażenie wielowyrazowe)

- kolejne nierozłączne wyrazy tworzące logiczne wyrażenie;
- realizowane przez:
 - przyimkowo – przymiotnikowe zwroty, np. po prostu, co gorsza;
 - przyimkowo – przysłówkowe zwroty, np. na pewno, na zewnątrz,
 - zwroty przysłówkowo – przyimkowe, np. wraz z, zgodnie z,
 - złożone spójniki, np. nie tylko, ale także, mimo że, o tyle – o ile,
 - związki przymiotnikowe, np. biało-czerwona.

pred (predykat podrzędny)

- realizowane przez formę czasownika lub przez główny rzeczownik w niezależnej frazie rzeczownikowej.

punct (znak interpunkcyjny)

- realizowany przez znak interpunkcyjny, np.: ?, !, (,), ,, ,, -.

abbrev_punct (znacznik skrótu)

- realizowany przez kropkę z poprzedzającym skrótem.

4.3.3 Typy nieargumentowe (motywowane morfologicznie)

aglt (aglutynat czasownika)

- realizowana przez „ruchomy“ affiks,
- zależy od formy czasownika lub warunkowej enklityki dołączanej do formy czasownika.

aux (czasownik posiłkowy)

- realizowany przez czasowniki pomocnicze „być“ lub „zostać“,
- zależy od formy czasownika głównego (bierny, bezokolicznikowy); w konstrukcjach analitycznych czasu przyszłego, w konstrukcjach warunkowych przeszłych, oraz konstrukcjach pasywnych.

cond (znacznik trybu przypuszczającego)

- realizowana przez partykulę „by“, która jest dołączana do formy czasownika lub pojawia się w dowolnym miejscu zdania.

imp (znacznik rozkazujący)

- realizowany przez partykulę „niech“,
- zależy od formy głównego czasownika w nadrzędnych konstrukcjach analitycznych.

neg (znacznik negacji)

- realizowany przez partykulę negującą „nie“,
- zależy od wyrażenia tuż po nim.

refl (znacznik zwrotności)

- realizowany przez partykulę „się“,
- zależy od czasownika w formie zwrotnej.

4.3.4 Typy koordynujące

conjunct (element koordynacji)

- zależy od połączenia koordynującego.

coord (spójnik współrzędny)

- koordynuje dwa zdania.

coord_punct (znak interpunkcji koordynowania)

- element koordynowania realizowany przez znak interpunkcyjny, np. :, ;.

pre_coord (element skorelowanego koordynowania)

- pierwszy człon w spójnikach skorelowanych, np. Albo ... albo

4.4 Przygotowanie modeli HMM

Ukryty model Markowa w naturalny sposób realizuje podejście oparte o model przejść, jak i w pewnym sensie dostarcza oceny zależności przez prawdopodobieństwo wystąpienia obserwacji względem stanu w modelu. W niniejszej pracy rozpatrywane są dwa języki polski i angielski, i dla obu tych języków utworzono osobne ukryte modele Markowa, których przestrzeń stanów oparta jest o zestaw znaczników wykorzystywany przez zbiory drzew składniowych.

4.4.1 Obserwacja zorientowana na afiksy

Jak to zostało przedstawione w pracy (Lioma 2008) użycie n -gramowej obserwacji pozwala na analizę bardziej złożonych struktur językowych. Wraz ze zwiększaniem rzędu n -gramowej obserwacji (liczby wyrazów w sekwencji będącej pojedynczą obserwacją), należy jednak dostarczyć odpowiednio większy zestaw danych uczących, aby wynik uczenia był znaczący statystycznie, niemniej dane treningowe nadal będą niewystarczające, aby pokryć wszystkie możliwe sekwencje wyrazów. Dlatego też struktura obserwacji w HMM powinna uwzględniać pewien stopień niepewności. Badanie przedstawione w pracy (Thede 1998, Nakagawa i in. 2001) wskazuje na niedostatek w klasyfikacji nowych instancji wyrazów, które nie były uwzględnione w bazie danych uczących. Skutki deficytu danych uczących na model mogą być do pewnego stopnia złagodzone, wykorzystując algorytmy wygładzające, opisane obszernie w pracy (Hazem & Morin 2013).

Alternatywą dla metody słownikowej, w której strukturę n -gramową tworzą całe wyrazy, jest rozkład morfologiczny. Zalety wykorzystania morfemów zostały przedstawione w pracy (Tseng i in. 2005, Luong i in. 2010). W niniejszej rozprawie do analizy syntaktycznej wybrane zostały jedynie morfemy brzegowe, dlatego struktura obserwacji oparta została o afiksy wyrazów z naiwnym założeniem stałej długości afiksów (szczegółowy opis wpływu tak dobranej obserwacji na wyniki klasyfikacji HMM przedstawiono w (Pietras 2017)). Założenie to prowadzi do utworzenia okna obserwacji (zarówno dla prefiksu, jak i sufiksu wyrazu), które można rozszerzyć (w wyniku czego obserwacja jest dokładniejsza), lub zwęzić (co prowadzi do generalizacji znaczenia statystycznego). Rozmiar okna obserwacji można dopasować indywidualnie do każdego z gramów obserwacji. Takie podejście zastosowane do analizy syntaktycznej wymaga przeprowadzenia klasyfikacji i ujednoznacznienia tekstu na podstawie cechy charakterystyczne badanego wyrażenia wraz z jego otoczeniem.

Wydażność i skuteczność HMM w dużym stopniu zależy od złożoności obserwacji i jakości bazy danych. W celu redukcji złożoności modelu (przez zmniejszenie wymiaru macierzy emisji), jak i by poprawić mechanizm rozpoznawania nowych wyrazów, wprowadzona została obserwacja afiksowa. Zakłada się, że dla rozpoznawania kategorii syntaktycznych afiksy wyrazu mają większe znaczenie niż jego rdzeń. Stąd, każde słowo jest reprezentowane przez swój prefiks i sufiks, a rdzeń może być pominięty. Zarówno prefiks, jak i sufiks mają znaki ustalonej długości. W konsekwencji, dla krótkich wyrazów wartości afiksów pokrywają się redundancyjnie (np. dla słowa „robi” afiksy długości 3 znaków to: rob-obi), natomiast dla długich wyrazów środkowa część wyrazu jest pominięta (np. dla słowa „pozyskuje” afiksy długości 3 znaków to: poz-uje). Jak widać, dużą zaletą tego podejścia jest natywny sposób inkluzji nowych wyrazów, niewystępujących podczas uczenia modelu. Ponadto, stała długość afiksów ogranicza możliwe kombinacje literowe, a co za tym idzie większa część kategorii syntaktycznych może zostać pokryta już przy niewielkich rozmiarach bazy uczącej. Innymi słowy, obserwacja zorientowana afiksowo grupuje wiele wyrazów w ramach pojedynczej obserwacji (np. wyrazy: **rozszerzać**, **rozmrażać**, **roztawiać** wyrażone są tą samą afiksową obserwacją). Wyrazy posiadające różne nieznanne rdzenie będą kategoryzowane według ich prefiksu i sufiksu, których znaczenie model poznał w procesie uczenia.

n -gramowa struktura afiksów

Podstawowa obserwacja (1-gramowa) tworzona jest z określonej długości afiksów wyrazów, natomiast w wariancie n -gramowym obserwacja tworzona jest z połączenia afiksów z n poprzednich wyrazów, które są przesuwane krocząco. Budowa n -gramowej obserwacji nie musi być

jednolita. Dla przeszłych (starszych) gramów obserwacji - morfemy brzegowe mogą być krótsze. Tym samym liczba kombinacji ulega zmniejszeniu. Ogólną idee przedstawia Tabela 4.1.

Wyraz	Unigram	Bigram	Trigram
those	tho-ose	___-___: tho-ose	___-___: ___-___: tho-ose
who	who-who	th-_se: who-who	___-___: th-_se: who-who
cannot	can-not	wh-_ho: can-not	t___-e: wh-_ho: can-not
remember	rem-ber	ca-_ot: rem-ber	w___-o: ca-_ot: rem-ber
the	the-the	re-_er: the-the	c___-t: re-_er: the-the
past	pas-ast	th-_he: pas-ast	r___-r: th-_he: pas-ast
are	are-are	pa-_st: are-are	t___-e: pa-_st: are-are
condemned	con-ned	ar-_re: con-ned	p___-t: ar-_re: con-ned
to	to-_to	co-_ed: to-_to	a___-e co-_ed: to-_to
repeat	rep-eat	to-_to: rep-eat	c___-d: to-_to: rep-eat
it	it-_it	re-_at: it-_it	t___-o: re-_at: it-_it

Tabela 4.1: Zastosowanie obserwacji afksowej w wariancie Uni-, Bi- i Tri-gramowym. (Źródło: *opracowanie własne*).

Ocena struktury afiksów

Podczas korzystania z obserwacji afksowej pojawia się pytanie o jakość odwzorowania. Jeśli długość afksów jest na tyle duża, że wszystkie wyrazy są bezpośrednio mapowane to wówczas nawet bardzo podobne wyrazy są rozróżnialne. Jeśli długość jest zbyt mała, zbyt wiele różnych wyrazów będzie grupowana i dokładność rozpoznawania składniowego ucierpi lub nawet spadnie do zera. Tabela 4.2 przedstawia zasięg słownika w odniesieniu do różnych długości afiksów.

Nr	Prefiks	Sufiks	Obserwacje dla j. angielskiego	Pokrycie słownika j. angielskiego [%]	Obserwacje dla j. polskiego	Pokrycie słownika j. polskiego [%]
1	1	1	685	1.07	967	1.03
2	1	2	5 191	8.12	5 879	6.29
3	1	3	18 152	28.40	18 952	20.29
4	1	4	35 291	55.22	36 264	38.82
5	1	5	48 180	75.39	54 232	58.06
6	2	1	4 518	7.06	6 681	7.15
7	2	2	17 710	27.71	20 469	21.91
8	2	3	36 086	56.46	39 447	42.23
9	2	4	49 322	77.17	56 025	59.98
10	2	5	56 689	88.70	70 834	75.84
11	3	1	20 479	32.04	25 185	26.96
12	3	2	38 939	60.93	46 657	49.95
13	3	3	51 251	80.19	62 991	67.44
14	3	4	57 689	90.27	74 076	79.31
15	3	5	61 065	95.55	83 169	89.04
16	4	1	40 348	63.13	49 335	52.82
17	4	2	52 309	81.85	68 175	72.99
18	4	3	58 367	91.33	78 218	83.74
19	4	4	61 268	95.87	84 347	90.31
20	4	5	62 773	98.22	89 550	95.88
21	5	1	52 345	81.90	67 430	72.19
22	5	2	58 654	91.78	80 607	86.30
23	5	3	61 535	96.28	86 370	92.47
24	5	4	62 784	98.24	89 962	96.32
25	5	5	63 423	99.24	92 882	99.44
Wszystkie wyrazy:			63 906	100	93 396	100

Tabela 4.2: Pokrycie słownika Wordnet w odniesieniu do różnych długości znaków w afiksach. (Źródło: *opracowanie własne*).

Jak ukazuje Tabela 4.2, długość afiksów wpływa na reprezentację i grupowanie wyrazów. Tabela ta powstała w aplikacji HMM-Toolbox w wyniku przypisywania struktury obserwacji dla słownika Wordnet-EN i Wordnet-PL (Maziarz i in. 2016). W języku angielskim dla prefiksu i sufiksu o długości 2 znaków każdy, klarowność reprezentacji słowa wynosi około 28% (17 710 różnych afiksów reprezentuje 63 906 wyrazów), natomiast dla afiksów o pojedynczym znaku klarowność reprezentacji słowa spada do 1% (685 różnych afiksów reprezentuje 63 906 słów). Podobna sytuacja występuje w języku polskim.

Dla zupełnie nowego wyrazu (nie ujętego w bazie danych uczących lub zawierającego błędy literowe) mogą być stosowane pewne dodatkowe metody inkluzji. Dla nowego wyrażenia możliwe jest określenie odległości Levenshteina (Levenshtein 1966) w celu znalezienia najbliższej obserwacji w przybliżeniu odpowiadającej temu wyrażeniu. Do obserwacji n -gramowej odległość Levenshteina powinna być obliczana osobno dla każdego grama obserwacji w n -gramowej sekwencji. Otrzymane wartości powinny być ważone tak, że najstarsze gramy posiadają najmniejszą wagę, a do najmłodszych gramów przypisywana jest najwyższa waga. Zależnie od analizowanego języka odległość Levenshteina może być zmodyfikowana w celu złagodzenia wpływu sufiksów lub prefiksów tak jak to przedstawiono w pracy (Pietras 2014b). Jeżeli natomiast wyrażenie nie pasuje do żadnej z obserwacji (w odległości Levenshteina różnica przekracza akceptowalny próg) to jest klasyfikowane jako nieznane (---).

4.4.2 Wybór modeli i danych uczących

W trakcie prowadzenia badań sprawdzone zostały różne kombinacje oznakowanych syntaktycznie tekstów w celu wyłonienia najkorzystniejszego podejścia. Dla języka angielskiego niewątpliwie najkorzystniej jest przeprowadzić uczenie modelu w oparciu o dobrze znany bank drzew Penn Treebank, który to ma ugruntowaną pozycję w dziedzinie lingwistyki komputerowej. Struktura banku zawiera oznakowanie danych tekstowych na różnym poziomie syntaktyczno-semantycznym, począwszy od oznakowania tekstu częściami mowy i częściami zdania, a kończąc na określeniu funkcji semantycznych. Dla języka polskiego wybrany został bank drzew Składnica, będący największym zbiorem oznakowanych syntaktycznie i semantycznie zdań w języku polskim. Ponadto, zaletą obu banków drzew jest ich szerokie rozpowszechnienie więc i możliwość porównania otrzymywanych wyników rozpoznawania.

4.5 Ocena modeli

W wynikach ilościowych raportowanych w dalszych częściach tego rozdziału dokładność działania poszczególnych modeli Markowa oceniona została za pomocą miar: precyzji, czułości oraz $F1$. Przy czym każda z tych miar była wyznaczana na rzecz różnych klas rozpoznawanych w ramach danego typu zadania. Klasy te są utożsamiane z odpowiednimi stanami w modelach Markowa.

Gwoli wyjaśnienia, *precyzja* określa odsetek obiektów danej klasy wśród wszystkich obiektów, które klasyfikator rozpoznał jako przynależne do tejże klasy. Sens probabilistyczny precyzji oddaje prawdopodobieństwo warunkowe $P(y(x) = \text{klasa} | c(x) = \text{klasa})$, gdzie $y(x)$ i $c(x)$ oznaczają odpowiednio prawdziwą klasę obiektu x oraz odpowiedź klasyfikatora dla tego obiektu. Miara *czułości* ma niejako odwrotny sens w rozumieniu prawdopodobieństwa warunkowego. Określa odsetek obiektów prawidłowo sklasyfikowanych do danej klasy wśród wszystkich obiektów faktycznie przynależnych do tejże klasy – czyli $P(c(x) = \text{klasa} | y(x) = \text{klasa})$. Miara $F1$ to średnia harmoniczna precyzji i czułości.

4.5.1 Model rozpoznawania kategorii syntaktycznych dla języka angielskiego

Kategorie syntaktyczne dla języka angielskiego przypisane do stanów w ukrytym modelu Markowa przedstawia Tabela 4.3.

Numer stanu	Nazwa stanu	Opis	Przykłady językowe
0	unknown	Kategoria nieznaną	„q#qhhssp\$%rr“
1	NN	Rzeczownik l. pojedynczej lub niepoliczalny	„tiger“, „chair“, „laughter“
2	IN	Koniunkcja podporządkowanie lub przyimek	„of“, „on“, „before“, „unless“
3	DT	Rodzajnik	„the“, „a“, „these“
4	VBZ	Czasownik l. pojedynczej 3. osoby, w czasie teraźniejszym	„she thinks“
5	RB	Przysłówek	„extremely“, „loudly“, „hard“
6	VDN	Trzecia forma czasownika, (imiesłów czynny)	„a sunken ship“
7	TO	Bezokolicznikowa forma „to“	„what to do?“
8	VB	Czasownik, forma podstawowa	„think“
9	JJ	Przymiotnik	„nice“, „easy“
10	NNS	Rzeczownik, l. mnoga	„tigers“, „chairs“, „insects“
11	NNP	Rzeczownik, nazwy własne	„Germany“, „God“, „Alice“
12	,	Znak interpunkcyjny, przecinek	„ , “
13	CC	Koniunkcja, wyrażenie koordynujące	„and“, „or“, „but“
14	POS	Dopełniacz dzierżawczy	„’s“
15	.	Znak interpunkcyjny zamykający zdanie	„ . “, „ ; “, „ ? “, „ * “
16	VBP	Czasownik, l. i 2. osoby, l. pojedynczej w czasie teraźniejszym	„I think“
17	VBG	Imiesłów	„thinking is fun“
18	PRP\$	Zaimek dzierżawczy	„my“, „your“, „our“
19	CD	Liczebnik główny	„five“, „three“, „13%“
20	‘ ’	Znak interpunkcyjny	‘ ’
21	„	Znak interpunkcyjny	„ , “
22	VBD	Czasownik w czasie przeszłym	„they thought“
23	EX	Egzystencjalna forma „there“	„there were six boys“
24	MD	Czasownik pomocniczy modalny	„may“, „should“
25	#	Znak specjalny	„ # “
26	(	Separator kontekstowy, lewy nawias	„(“
27	\$	Znak specjalny	„ \$ “
28	)	Separator kontekstowy, prawy nawias	„)“
29	NNPS	Rzeczownik, odpowiednia l. mnoga	„we met two Christmases ago“
30	PRP	Zaimek osobowy	„me“, „you“, „it“
31	JJS	Przymiotnik superlatywny	„nicest“, „easiest“
32	WP	Zaimek osobowy co“, „kto“, „komu	„what“, „who“, „whom“
33	RBR	Przysłówek porównawczy	„better“
34	JJR	Przymiotnik porównawczy	„nicer“, „easier“
35	WDT	Wyznacznik, cokolwiek, którykolwiek	„which“, „whatever“, „whichever“
36	WRB	Przysłówek gdzie, kiedy	„where“, „when“
37	RBS	Przysłówek superlatywny	„best“
38	PDT	Wyznacznik poprzedzający	„both his children“
39	RP	Przysłówek, partykuła	„about“, „off“, „up“
40	:	Znak interpunkcyjny	„ : “
41	FW	Wyraz obcy	„mais“
42	WP\$	Zaimek dzierżawczy od „kto“	„whose“, „whosever“
43	SYM	Symbol	„%“
44	UH	Wykrzyknik	„oh“, „oops“, „gosh“

Tabela 4.3: Kategorie syntaktyczne dla języka angielskiego przypisane do stanów w ukrytym modelu Markowa.

(Źródło: opracowane na podstawie (Santorini i in. 1989)).

Wynik rozpoznawania kategorii syntaktycznych dla języka angielskiego

W wyniku uczenia określonych zostało 17 027 różnych obserwacji. Ocena modelu na zestawie testowym wykazała, że dla około 6.19% (tj. 2 934 spośród 47 367) wyrażeń nie udało się zidentyfikować pasującej obserwacji, stąd zostały one przypisane do kategorii „nieznana”.

Numer stanu	Nazwa stanu	Precyzja dla znanych [%]	Precyzja dla nieznanych [%]	Precyzja wszystkich [%]	Czułość wszystkich [%]	Miara F1 wszystkich [%]
0	unknown	-	-	-	-	-
1	NN	92.34	49.46	89.40	86.01	87.67
2	IN	98.23	-	98.23	98.63	98.43
3	DT	99.53	-	99.53	99.15	99.34
4	VBZ	96.18	15.39	94.98	91.77	93.35
5	RB	95.69	-	95.69	90.03	92.77
6	VBN	87.17	41.00	83.10	81.32	82.20
7	TO	99.83	-	99.83	100	99.91
8	VB	94.45	69.74	92.95	88.02	90.41
9	JJ	91.10	39.93	82.30	81.53	81.91
10	NNS	96.25	72.73	96.15	93.18	94.64
11	NNP	92.16	68.85	86.58	90.92	88.70
12	.	99.96	-	99.96	100	99.98
13	CC	99.76	-	99.76	99.75	99.75
14	POS	99.31	-	99.31	98.84	99.07
15	.	99.90	-	99.90	100	99.95
16	VBP	93.14	-	93.14	89.82	91.45
17	VBG	93.90	-	93.90	94.85	94.37
18	PRP\$	100	-	100	99.52	99.76
19	CD	100	41.57	84.77	84.20	84.48
20	' '	100	-	100	100	100
21	..	100	-	100	99.36	99.68
22	VBD	94.53	34.89	92.98	91.89	92.43
23	EX	100	-	100	100	100
24	MD	100	-	100	97.23	98.59
25	#	100	-	100	100	100
26	(	100	-	100	100	100
27	\$	100	-	100	99.47	99.73
28	)	100	-	100	100	100
29	NNPS	79.75	-	79.75	70.70	74.95
30	PRP	99.27	-	99.27	99.63	99.44
31	JJS	92.31	-	92.31	93.50	92.90
32	WP	95.09	-	95.09	100	97.48
33	RBR	94.64	-	94.64	81.42	87.53
34	JJR	90.39	-	90.39	96.53	93.35
35	WDT	85.72	-	85.72	94.55	89.91
36	WRB	-	-	-	-	-
37	RBS	100	-	100	100	100
38	PDT	100	-	100	100	100
39	RP	100	-	100	100	100
40	:	100	-	100	100	100
41	FW	100	-	100	100	100
42	WP\$	100	-	100	100	100
43	SYM	-	-	-	-	-
44	UH	-	-	-	-	-
Razem		96.60	48.17	95.60	94.92	95.22

Tabela 4.4: Dokładność rozpoznawania kategorii syntaktycznych z wykorzystaniem ukrytego modelu Markowa dedykowanego dla języka angielskiego.

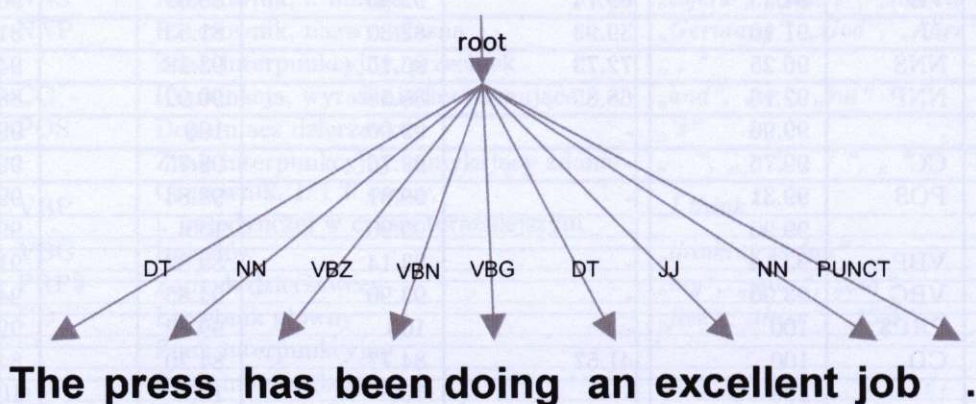
(Źródło: opracowanie własne).

Precyzja rozpoznawania obserwacji nieznanych wyniosła 48.17%, natomiast dla znanych obserwacji precyzja wyniosła już 96.60%, tym samym całkowita precyzja klasyfikacji dla wszystkich obserwacji w zestawie testowym wyniosła 95.60%. Całkowita czułość klasyfikacji dla wszystkich obserwacji w zestawie testowym wyniosła 94.92%. Stąd, miara F1 dla przedstawionego modelu wyniosła 95.22%.

Wybrane przykłady rozpoznawania kategorii syntaktycznych dla języka angielskiego

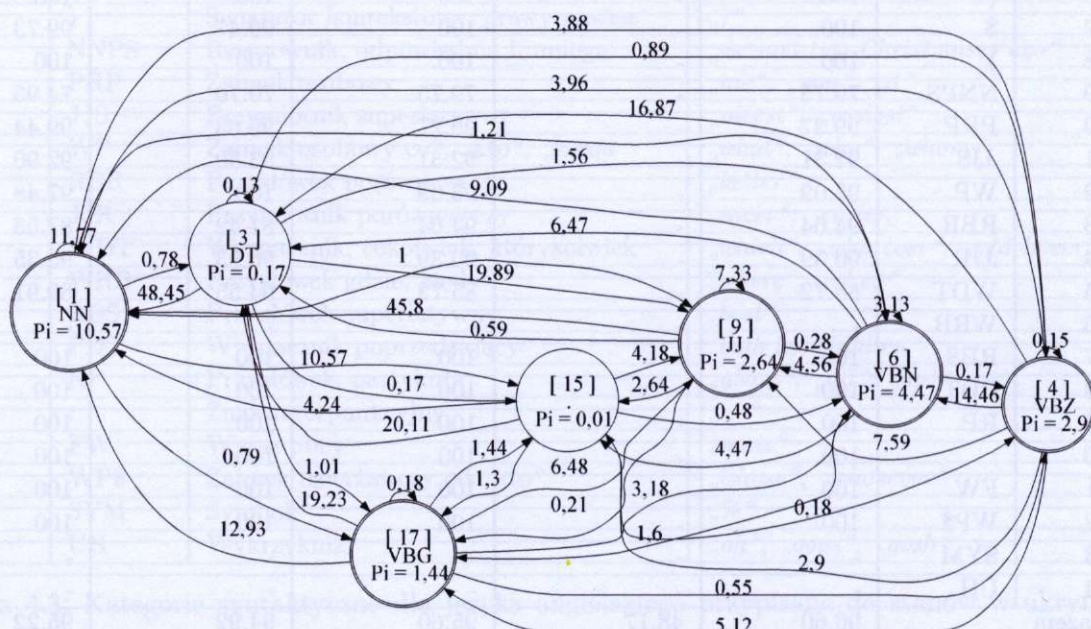
Ze względu na rozbudowaną strukturę modelu Markowa służącego do rozpoznawania kategorii syntaktycznych, jego graficzna prezentacja (patrz Rysunek 4.5) ograniczona została tu jedynie do stanów wykorzystanych w przykładowym zdaniu. Niemniej jednak należy pamiętać, że całkowita przestrzeń stanów jest dużo większa. Przykład zdania w języku angielskim:

„The press has been doing an excellent job.“



Rysunek 4.4: Przykład rozpoznawania kategorii syntaktycznych w zdaniu z wykorzystaniem ukrytego modelu Markowa.

(Źródło: opracowanie własne).



Rysunek 4.5: Model Markowa prezentujący prawdopodobieństwa przejść między wybranymi kategoriami syntaktycznymi.

(Źródło: opracowanie własne).

4.5.2 Model rozpoznawania grup frazowych dla języka angielskiego

Grupy wyrazów, które są powiązane ze sobą tworząc pewne spójne wyrażenie nazywamy frazą. Kategoria frazy zależy od występujących w niej części mowy. Najczęstszymi frazami są fraza orzeczownikowa (NP, na przykład „the red car“) i fraza czasownikowa (VP, na przykład „is broken“). Kategorie grup frazowych przypisane do stanów w ukrytym modelu Markowa przedstawia Tabela 4.5.

Numer stanu	Nazwa stanu	Opis	Części mowy	Przykłady językowe
0	unknown	Kategoria nieznana	{}	„q#qhhssp\$%rr“
1	B-NP	Rozpoczyna frazę rzeczownikową	DT+RB+JJ+NN+PR	„international business day“
2	B-PP	Rozpoczyna frazę przymkową	TO+IN	„because of“, „against“, „instead of“, „for“
3	I-NP	Wyrażenie wewnątrz frazy rzeczownikowej	DT+RB+JJ+NN+PR	„international business day“
4	B-VP	Rozpoczyna frazę czasownikową	RB+MD+VB	„to provide“, „takes“, „would have realized“
5	I-VP	Wyrażenie wewnątrz frazy czasownikowej	RB+MD+VB	„to provide“, „would have realized“
6	B-SBAR	Rozpoczyna frazę podporządkowaną	IN	„while“, „as if“, „although“
7	O	Wyrażenie lub znak niebędący frazą	.	„“, „?“, „!“
8	B-ADJP	Rozpoczyna frazę przymiotnikową	CC+RB+JJ	„poorly positioned“, „at least risky“, „higher“
9	B-ADVP	Rozpoczyna frazę przysłówkową	RB	„therefore“, „previously“, „once again“, „any further“
10	I-ADVP	Wyrażenie wewnątrz frazy przysłówkowej	RB	„once again“, „no longer“
11	I-ADJP	Wyrażenie wewnątrz frazy przymiotnikowej	CC+RB+JJ	„increasingly successful“, „more attractive“
12	I-SBAR	Wyrażenie wewnątrz frazy podporządkowanej	IN RB	
13	I-PP	Wyrażenie wewnątrz frazy przymkowej	TO+IN	„because of“, „instead of“, „rather than“
14	B-PRT	Rozpoczyna frazę partykuły	RP	„out“, „off and on“
15	B-LST	Rozpoczyna frazę znacznika listy	CD	„1“, „2“
16	B-INTJ	Rozpoczyna frazę wykrzyknika	UH	„beat, beat“, „wow“
17	I-INTJ	Wyrażenie wewnątrz frazy wykrzyknika	UH	„oh no“, beat, beat“
18	B-CONJP	Rozpoczyna frazę koniunkcji	RB	„not only“, „as well as“
19	I-CONJP	Wyrażenie wewnątrz frazy koniunkcji	RB	„not only“, „as well as“
20	I-PRT	Wyrażenie wewnątrz frazy partykuły	RP	„off and on“
21	B-UCP	Rozpoczyna frazę nietypowo skoordynowaną	CC	„wine, spirits and soft-drink“
22	I-UCP	Wyrażenie wewnątrz frazy nietypowo skoordynowanej	CC	„wine, spirits and soft-drink“
23	I-LST	Wyrażenie wewnątrz frazy znacznika listy	CD	„1 - 2“, „2 - 3“

Tabela 4.5: Kategorie grup frazowych dla języka angielskiego przypisane do stanów w ukrytym modelu Markowa.

(Źródło: opracowane na podstawie (Sang i in. 2000)).

samym całkowita precyzja klasyfikacji dla wszystkich obserwacji w zestawie testowym wyniosła 93.17%. Całkowita czułość klasyfikacji dla wszystkich obserwacji w zestawie testowym wyniosła 93.93%. Stąd, miara F1 dla klasyfikacji wybranych grup frazowych (NP+PP+VP+SBAR) wyniosła 93.55%.

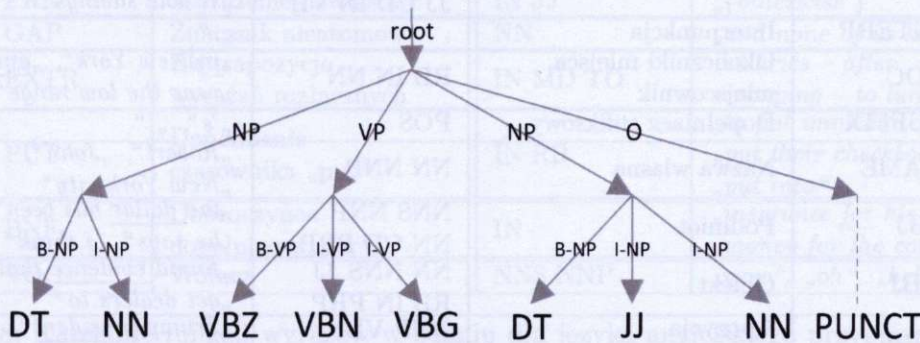
		Warstwowy ukryty model Markowa, dla którego obserwacje stanowią kombinacje afiksów wyrazów i odkrytych dla nich kategorii syntaktycznych				
Numer stanu	Nazwa stanu	Precyzja dla znanych [%]	Precyzja dla nieznanych [%]	Precyzja wszystkich [%]	Czułość wszystkich [%]	Miara F1 wszystkich [%]
0	Others	75.57	-	75.57	72.53	74.02
1	NP	97.75	88.22	97.07	95.77	96.41
2	PP	92.08	-	92.08	94.70	93.37
3	VP	92.21	-	92.21	91.81	92.01
4	SBAR	91.31	-	91.31	93.46	92.37
5	O	90.61	-	90.61	97.08	93.73
NP+PP+VP+SBAR		93.34	88.22	93.17	93.93	93.55
Razem		89.92	88.22	89.81	90.89	90.34

Tabela 4.7: Dokładność rozpoznawania grup frazowych z wykorzystaniem warstwowego ukrytego modelu Markowa dedykowanego dla języka angielskiego.
(Źródło: opracowanie własne).

Wybrane przykłady rozpoznawania grup frazowych dla języka angielskiego

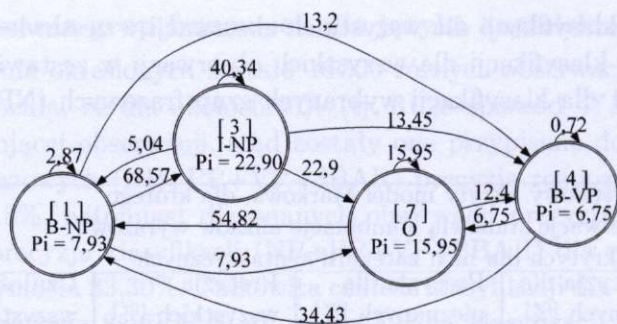
Ze względu na rozbudowaną strukturę modelu Markowa służącego do rozpoznawania grup frazowych, jego graficzna prezentacja (patrz Rysunek 4.7) ograniczona została tu jedynie do stanów wykorzystanych w przykładowym zdaniu. Niemniej jednak należy pamiętać, że całkowita przestrzeń stanów jest dużo większa. Przykład zdania w języku angielskim:

„The press has been doing an excellent job.“



The press has been doing an excellent job .

Rysunek 4.6: Przykład rozpoznawania grup frazowych w wykorzystaniem ukrytego modelu Markowa.
(Źródło: opracowanie własne).



Rysunek 4.7: Model Markowa prezentujący prawdopodobieństwa przejść między wybranymi grupami frazowymi.

(Źródło: opracowanie własne).

4.5.3 Model rozpoznawania funkcji wyrazów w zdaniu dla języka angielskiego

Przedstawione części zdania opisują relacje między różnymi wyrazami i wyjaśniają ich funkcje w zdaniu. Kluczowymi funkcjami w zdaniu są podmiot (SBJ) i orzeczenie (OBJ). Podmiotem w zdaniu może być osoba lub rzecz, która wykonuje czynność lub jest opisywana. Orzeczeniem w zdaniu jest działanie przypisane podmiotowi. Kategorie funkcji wyrazów w zdaniu dla języka angielskiego przypisane do stanów w ukrytym modelu Markowa przedstawia Tabela 4.8.

Numer stanu	Nazwa stanu	Opis	Grupy frazowe	Przykłady językowe
0	unknown	Kategoria nieznana	{}	
1	ADV	Okolicznik przysłówkowy	IN RB NN	„sharply“, „equivalent“, „including“, „wondering“
2	NMOD	Przydawka określająca rzeczownik	PRP JJ IN CD VB DT NN	„kingdom of great britain“, „shopping centers“
3	PMOD	Modyfikator przyimkowy	NN NNP NNS JJ CD IN VB	„black monday“, „sizable savings“
4	INTERP	Interpunkcja	,	„“
5	LOC	Okoliczniki miejsca, miejscownik	RB IN NN	„in New York“, „among other“, „near the low range“
6	SUFFIX	Dopełniacz sufiksowy	POS	„s“, „“
7	NAME	Nazwa własna	NN NNP	„Robert“, „John“, „New York city“
8	SBJ	Podmiot	NNS NNP NN CD PRP	„the dollar has been“, „he says“, „i think“
9	OBJ	Obiekt	NN NNS JJ RB IN PRP	„found evidence that“, „get dealers to“
10	APPO	Apozycja, figura retoryczna	NNP VBN NN JJ CD	„Trump president“, „apple company“
11	LGS	Podmiot logiczny dla biernej formy czasownika	IN	„compiled by“, „sold by“, „rattled by“
12	ROOT	Korzeń zdania	VBZ VBD MD VBP VBG	„they averaged about“, „suggested“, „risks“, „administration says“
13	AMOD	Modyfikator przyimkowy lub przysłówkowy	DT IN VBZ JJ NN IN RB TO RBR	„western european“, „more recently“, „relatively flat“
14	PRD	Uzupełnienie predykatu	JJ IN VBG VBN VBZ NN	„are less reliant“, „industry are skeptical“, „were different“
15	PUNCT	Interpunkcja zamykająca zdanie	.	„“
16	TITLE	Tytuł	NNP NN	„mr.“, „president“, „prof.“
17	HYPH	Znacznik będący łącznikiem wyrazów	-	„thirty - five“, „t - shirts“, „per - capita“

18	POSTHON	Modyfikator po grzecznościowy	NNP	„compaq computer corp.“, „toshiba corp.“
19	OPRD	Predykatyw dla czasownika	NN NNS TO VB	„trying to“, „began buying“
20	IM	Czasownik w formie bezokolicznikowej	VB	„to work toward“, „to try to get them“
21	TMP	Okolicznik czasu	RB IN NNP WRB NN VBD	„previously“, „within the next“, „from august“, „may soon be“
22	DEP	Nieokreślone relacje	CD TO NN JJ	„yet more than one“, „. But“
23	COORD	Spójnik	NNS VBD CC VBP VBZ	and“, „or“, „as“, „nor“, „but“
24	CONJ	Druga koniunkcja	VBD NNS JJ NN VB VBP NNP IN VBG	„company and union officials“, „science and technology“
25	DIR	Okolicznik kierunkowy	TO IN RB	„moved ahead“, „wedged into her“
26	VC	Łańcuch czasowników	VB VBN VBG	„were signed“, „is leaving“
27	PRP	Okolicznik celu, okolicznik przyczyny	IN TO	„because he may“, „to kick off“, „to distinguish itself“
28	HMOD	Wyrażenie wewnątrz wyrazu dzielonego	NN DT CD JJ IN RB	„head - to“, „cable - tv“, „prime - time“
29	SUB	Wyrażenie podrzędne	MD VBP VBZ VBD	„is totally“, „were sufficient“, „may be required“
30	MNR	Okolicznik sposobu	RB IN	„significantly exceed“, „sitting quietly“, „work together“
31	EXT	Okolicznik miary	IN	„nearly 200 points“, „deficit by more“, „100 tons“
32	DTV	Celownik uzupełniający „to“	TO	„offered to“, „paid to them“
33	PRT	Partykuła	RO	„an up day“, „heading down into“, „go down“, „pointed out“
34	P	Interpunkcja inna	,	„ „ „ „
35	PRN	Wyrażenie nawiasowe	IN JJ	„{otherwise}“
36	GAP	Znacznik nieatomowy	NN	„and none of us knows“
37	EXTR	Ekstrapozycja wyrażen rozłącznych	IN MD TO	„salaries – after scrimping – to lure“
38	PUT	Dopełnienie czasownika „put“	IN RB	„to put immediate heat on“, „put their checkbooks in“, „put into“
39	BNF	Dobroczyńca dopełniacza „for“	IN	„insurance for his girlfriend“, „money for the company“
40	VOC	Wołacz	NNS NNP	„wow“, „oh“, „hello“

Tabela 4.8: Kategorie funkcji wyrazów w zdaniu dla języka angielskiego przypisane do stanów w ukrytym modelu Markowa.

(Źródło: opracowane na podstawie (Surdeanu i in. 2008)).

Wynik rozpoznawania funkcji wyrazów w zdaniu dla języka angielskiego

W wyniku uczenia określonych zostało 37 368 różnych obserwacji. Ocena modelu na zestawie testowym wykazała, że dla około 1.92% (tj. 1 105 spośród 57 552) wyrażen nie udało się zidentyfikować pasującej obserwacji, stąd zostały one przypisane do kategorii „nieznana“. Precyzja rozpoznawania obserwacji nieznanych wyniosła 53.85%, natomiast dla znanych obserwacji precyzja wyniosła już 76.34%, tym samym całkowita precyzja klasyfikacji dla wszystkich obserwacji w zestawie testowym wyniosła 74.23%. Całkowita czułość klasyfikacji dla wszystkich obserwacji w zestawie testowym wyniosła 75.31%. Stąd, miara F1 dla przedstawionego klasyfikatora wyniosła 73.88%.

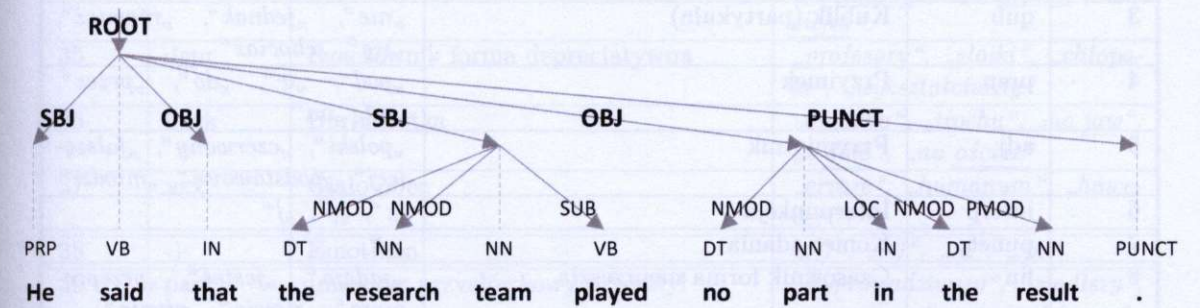
		Warstwowy ukryty model Markowa, dla którego obserwacje stanowią kombinacje afiksów wyrazów i odkrytych dla nich kategorii syntaktycznych				
Numer stanu	Nazwa stanu	Precyzja dla znanych [%]	Precyzja dla nieznanymi [%]	Precyzja wszystkich [%]	Czułość wszystkich [%]	Miara F1 wszystkich [%]
0	unknown	-	-	-	-	-
1	ADV	65.50	-	65.50	68.19	66.81
2	NMOD	81.44	54.93	80.97	82.03	81.49
3	PMOD	52.12	39.82	51.63	69.17	59.13
4	INTERP	92.81	-	92.81	86.27	89.42
5	LOC	53.93	-	53.93	86.56	66.45
6	SUFFIX	98.71	-	98.71	100	99.35
7	NAME	69.50	63.27	69.22	88.83	77.81
8	SBJ	72.30	70.47	72.22	71.28	71.74
9	OBJ	78.47	-	78.47	74.70	76.54
10	APPO	64.79	-	64.79	43.01	51.70
11	LGS	62.11	-	62.11	91.46	73.98
12	ROOT	52.66	32.43	52.43	70.06	59.97
13	AMOD	66.17	-	66.17	57.14	61.32
14	PRD	74.82	-	74.82	52.90	61.97
15	PUNCT	85.31	-	85.31	90.03	87.60
16	TITLE	94.23	-	94.23	94.09	94.16
17	HYPH	99.01	-	99.01	97.51	98.25
18	POSTHON	90.46	-	90.46	95.06	92.70
19	OPRD	51.88	-	51.88	68.71	59.12
20	IM	93.85	66.67	93.13	96.98	95.01
21	TMP	76.41	-	76.41	52.14	61.98
22	DEP	69.86	57.53	69.10	61.84	65.27
23	COORD	88.35	-	88.12	75.01	81.04
24	CONJ	79.83	63.16	78.96	65.15	71.39
25	DIR	-	18.49	18.49	41.13	25.51
26	VC	82.40	41.67	81.69	91.54	86.33
27	PRP	79.59	-	79.59	70.62	74.83
28	HMOD	85.27	62.50	84.77	84.51	84.64
29	SUB	-	-	-	-	-
30	MNR	-	-	-	-	-
31	EXT	-	-	-	-	-
32	DTV	-	-	-	-	-
33	PRT	91.28	-	91.28	97.51	94.29
34	P	-	-	-	-	-
35	PRN	66.67	-	66.67	35.89	46.66
36	GAP	-	-	-	-	-
37	EXTR	-	-	-	-	-
38	PUT	-	-	-	-	-
39	BNF	-	-	-	-	-
40	VOC	-	-	-	-	-
Razem		76.34	51.90	74.23	75.31	74.88

Tabela 4.9: Dokładność rozpoznawania funkcji wyrazów w zdaniu z wykorzystaniem ukrytego modelu Markowa dedykowanego dla języka angielskiego.
(Źródło: opracowanie własne).

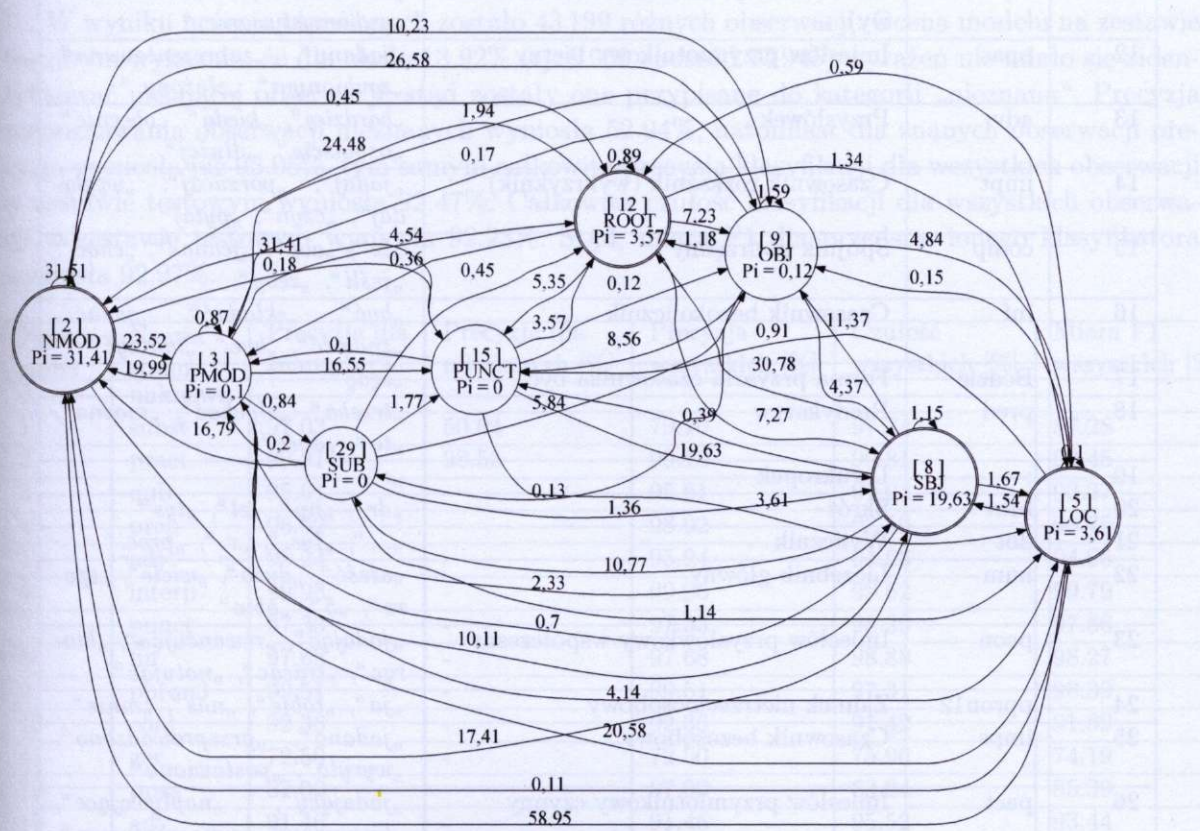
Wybrane przykłady rozpoznawania funkcji wyrazów w zdaniu dla języka angielskiego

Ze względu na rozbudowaną strukturę modelu Markowa służącego do rozpoznawania funkcji wyrazów w zdaniu, jego graficzna prezentacja (patrz Rysunek 4.9) ograniczona została tu jedynie do stanów wykorzystanych w przykładowym zdaniu. Niemniej jednak należy pamiętać, że całkowita przestrzeń stanów jest dużo większa. Przykład zdania w języku angielskim:

„He said that the research team played no part in the result.“



Rysunek 4.8: Przykład rozpoznawania funkcji wyrazów w zdaniu z wykorzystaniem ukrytego modelu Markowa.
(Źródło: opracowanie własne).



Rysunek 4.9: Model Markowa prezentujący prawdopodobieństwa przejść między wybranymi kategoriami funkcji wyrazów w zdaniu.
(Źródło: opracowanie własne).

4.5.4 Model rozpoznawania kategorii syntaktycznych dla języka polskiego

Kategorie syntaktyczne dla języka polskiego przypisane do stanów w ukrytym modelu Markowa określone w (Przepiórkowski i in. 2012) jako klasy fleksemów przedstawia Tabela 4.10.

Numer stanu	Nazwa stanu	Opis	Przykłady językowe
0	unknown	Kategoria nieznana	{ }
1	subst	Rzeczownik	„kot“, „profesorowie“, „godziny“, „komitetu“, „pieniądze“
2	praet	Czasownik pseudoimiesłowowy	„jadał“, „stał“, „miał“, „zranił“, „doprowadziło“
3	qub	Kublik (partykuła)	„nie“, „jednak“, „również“, „się“, „chociaż“
4	prep	Przyimek	„pod“, „w“, „do“, „przez“, „o“, „na“
5	adj	Przymiotnik	„polski“, „czerwony“, „fałszywej“, „podstawowe“, „młodzi“
6	interp	Interpunkcja	„;“, „(“, „)“
7	punct	Koniec zdania	„.“
8	fin	Czasownik forma nieprzeszła	„jadam“, „jesteś“, „przepraszam“, „zjawia“, „płynie“
9	ppron3	Zaimek trzecioosobowy	„on“, „jemu“, „jej“, „jego“, „go“
10	conj	Spójnik współrzędny	„oraz“, „lub“, „lecz“, „i“, „jednak“
11	ger	Czasownik odsłownik (rzeczownik odczasownikowy, imiesłów odczasownikowy)	„jadanie“, „malowanie“, „zwiedzanie“, „dokonaniu“, „wykorzystywanie“
12	ppas	Imiesłów przymiotnikowy bierny	„jadany“, „zaprezentowana“, „zgubionym“, „złożony“
13	adv	Przysłówek	„bardziej“, „kiedy“, „obecnie“, „osobiście“, „dłużej“
14	impt	Czasownik rozkaznik (wykrzyknik)	„jadaj“, „porzucaj“, „wyglądaj“, „czyn“, „pytaj“
15	comp	Spójnik podrzędny	„że“, „aby“, „jednak“, „choć“, „jeśli“, „żeby“
16	inf	Czasownik bezokolicznik	„być“, „składać“, „jadać“, „znaleźć“, „być“, „marzyć“
17	Bedzie	Forma przyszła czasownika być	„będę“
18	pred	Predykatyw	„trzeba“, „słychać“, „można“, „to“, „dość“
19	:	Dwukropek	„.“
20	brev	Skrót	„dr“, „np“, „zł“, „tys“
21	dot	Wylicznik	„...“, „tys.“, „r.“, „proc.“
22	num	Liczebnik główny	„sześć“, „dużo“, „wiele“, „sporo“, „5“, „obie“
23	pcon	Imiesłów przysłówkowy współczesny	„jadając“, „rewanżując“, „biorąc“, „tracąc“, „notując“
24	ppron12	Zaimek nietrzecioosobowy	„ja“, „tobie“, „nas“, „mnie“
25	imps	Czasownik bezosobowy	„jadano“, „przeprowadzono“, „wryto“, „postawiono“
26	pact	Imiesłów przymiotnikowy czynny	„jadający“, „napływające“, „kierujący“, „umacniającym“, „wygrzewający“
27	aglt	Aglutynant czasownika być	„chciał -em“, „by -m“, „sprawdzili -śmy“
28	interj	Wykrzyknik fatyczny (emotikon)	„ach“, „psiakrew“, „o kurczę“, „precz“

29	numcol	Liczebnik zbiorowy	„sześcioro“, „trojga“, „dwojgiem“
30	siebie	Siebie	„sobą“, „siebie“
31	adjc	Przymiotnik predykatywny	„wesół“, „pewien“, „gotów“
32	winien	Czasownik typu winien (forma terażniejsza)	„winna“, „powinni“
33	adja	Przymiotnik przy-przymiotnikowy	„polsko-ukraińska“, „chrześcijańsko-narodowy“, „warmińsko-mazurski“
34	adjp	Przymiotnik po-przyminkowy	„po polsku“, „po angielsku“, „od dawna“, „po prostu“, „po cichu“
35	depr	Rzeczownik forma deprecjatywna	„profesory“, „słoiki“, „chłopa-ki“, „wysztalciuchy“
36	burk	Burkinostka	„omacku“, „trochu“, „na jaw“, „z dala“, „na oścież“
37	xxx	Ciało obce	„errare“, „humanum“, „however“
38	-)	Emotikon	„:-)“, „;-)“, „:d“, „:p“,
39	pant	Imiesłów przysłówkowy przeszły!	„wyprowadziwszy“, „zjadłszy“, „poznałszy“, „stwierdziwszy“

Tabela 4.10: Kategorie syntaktyczne dla języka polskiego przypisane do stanów w ukrytym modelu Markowa.

(Źródło: opracowane na podstawie (Przepiórkowski i in. 2012)).

Wynik rozpoznawania kategorii syntaktycznych dla języka polskiego

W wyniku uczenia określonych zostało 43 199 różnych obserwacji. Ocena modelu na zestawie testowym wykazała, że dla około 13.92% (tj. 9 969 spośród 73 945) wyrażeń nie udało się zidentyfikować pasującej obserwacji, stąd zostały one przypisane do kategorii „nieznana“. Precyzja rozpoznawania obserwacji nieznanych wyniosła 52.94%, natomiast dla znanych obserwacji precyzja wyniosła już 95.05%, tym samym całkowita precyzja klasyfikacji dla wszystkich obserwacji w zestawie testowym wyniosła 92.47%. Całkowita czułość klasyfikacji dla wszystkich obserwacji w zestawie testowym wyniosła 92.23%. Stąd, miara F1 dla przedstawionego klasyfikatora wyniosła 92.97%.

Numer stanu	Nazwa stanu	Precyzja dla znanych [%]	Precyzja dla nieznanych [%]	Precyzja wszystkich [%]	Czułość wszystkich [%]	Miara F1 wszystkich [%]
0	unknown	-	-	-	-	-
1	subst	97.03	50.04	79.10	97.34	87.28
2	praet	99.21	98.53	99.19	99.81	99.45
3	qub	95.61	-	95.61	91.04	93.27
4	prep	98.92	-	98.92	98.59	98.75
5	adj	95.24	-	95.24	94.66	94.95
6	interp	99.98	-	99.96	99.62	99.79
7	punct	97.33	-	97.33	98.39	97.86
8	fin	97.68	-	97.68	98.88	98.27
9	ppron3	99.51	-	99.51	97.31	98.39
10	conj	92.36	-	92.36	91.42	91.89
11	ger	72.50	-	72.50	75.96	74.19
12	ppas	87.00	-	87.00	83.84	85.39
13	adv	91.46	-	91.46	95.52	93.44
14	impt	100	-	100	88.73	94.03
15	comp	90.69	-	90.69	96.04	93.29
16	inf	99.01	-	99.01	99.01	99.01
17	Bedzie	100	-	100	99.37	99.68
18	pred	61.47	-	58.03	78.88	66.86

19	:	100	-	100	100	100
20	brev	95.77	-	95.77	97.29	96.52
21	dot	86.30	-	86.30	84.25	85.26
22	num	90.04	-	81.30	87.29	84.18
23	pcon	99.08	-	99.08	97.29	98.17
24	ppron12	87.50	-	87.50	97.10	92.05
25	imps	100	-	100	100	100
26	pact	95.49	-	95.49	95.00	95.24
27	aglt	99.50	-	99.50	97.52	98.50
28	interj	92.31	-	92.31	51.85	66.40
29	numcol	100	-	100	98.88	99.43
30	siebie	100	-	100	99.27	99.63
31	adjc	100	-	100	100	100
32	winien	100	-	100	98.00	98.99
33	adja	93.34	-	93.34	93.33	93.33
34	adjp	92.86	-	92.86	86.47	89.55
35	depr	100	-	100	100	100
36	burk	100	50.00	72.50	62.5	67.13
37	xxx	100	13.21	57.02	66.7	61.48
38	-)	100	-	100	100	100
39	pant	100	-	100	100	100
Razem		95.05	52.94	92.47	92.23	92.97

Tabela 4.11: Dokładność rozpoznawania kategorii syntaktycznych z wykorzystaniem ukrytego modelu Markowa dedykowanego dla języka polskiego.

(Źródło: *opracowanie własne*).

Model rozpoznawania kategorii syntaktycznych z podziałem na grupy gramatyczne dla języka polskiego

Kategorie gramatyczne przyjęte w NKJP (Narodowym Korpusie Języka Polskiego) tworzą liczny zbiór kombinacji (Przepiórkowski i in. 2012). Uwzględniając wszystkie przysługujące poszczególnym fleksemom kombinacje, możliwe jest rozróżnienie 763 kategorii syntaktycznych. Przy tak licznym zestawie pojawia się problem wielkości bazy danych uczących. Przykładowo dany wyraz będący rzeczownikiem może być odmieniony przez przypadki (7), rodzaje (5) i liczbę (2), co pozwala na zamapowanie go do 70 kategorii. Dlatego też zaleca się dopasowanie zestawu kategorii w zależności od potrzeb i utworzenie nie jednego, a kilku wyspecjalizowanych ukrytych modeli Markowa. Przyjęto następujące specjalizacje dla ukrytych modeli Markowa:

- wyspecjalizowany w rozpoznawaniu przypadków, osoby, przymiotnikowości i aspektu,
- wyspecjalizowany w rozpoznawaniu liczby, rodzaju, stopnia i akomodacyjności,
- wyspecjalizowany w rozpoznawaniu negacji, akcentowości, aglutynacyjności i wokaliczności.

Kategorie przypisane do stanów w ukrytym modelu Markowa rozszerzonym o rozpoznawanie przypadków, osób, przymiotnikowości i aspektu przedstawia Tabela 4.12.

Modyfikator stanu	Opis	Przykład obserwacji
Przypadek modyfikuje kategorie: subst depr num numcol adj ppron12 ppron3 siebie ger pact ppas		
nom	mianownik	koza, morze, ucho
gen	dopełniacz	kozy, morza, ucha
dat	celownik	kozie, morzu, uchu
acc	biernik	kozę, morze, ucho
inst	narzędnik	kozą, morzem, uchem
loc	miejscownik	kozie, morzu, uchu
voc	wołacz	kozy, morze, ucho
Osoba modyfikuje kategorie: ppron12 ppron3 fin Bedzie aglt impt		
prl	Pierwsza osoba	jadę, pływam, gram
sec	Druga osoba	jedziesz, pływasz, grasz
ter	Trzecia osoba	jedzie, pływa, gra
Przymiotnikowość modyfikuje kategorię: ppron3		
praep	poprzyimkowa	niego, -n
npraep	nipoprzyimkowa	jego, go
Aspekt modyfikuje kategorie: fin Bedzie aglt praet impt pact ppas winien		
imperf	Czas niedokonany	jeść, słuchać, mówić
perf	Czas dokonany	zjeść, zasłyczeć, zrobić

Tabela 4.12: Kategorie syntaktyczne rozszerzone o rozpoznawanie przypadków, osoby, przymiotnikowości i aspektu dla języka polskiego przypisane do stanów w modelu Markowa. (Źródło: *opracowane na podstawie (Przepiórkowski i in. 2012)*).

Kategorie przypisane do stanów w ukrytym modelu Markowa rozszerzonym o rozpoznawanie liczby, rodzaju, stopnia i akomodacyjności, przedstawia Tabela 4.13.

Modyfikator stanu	Opis	Przykład obserwacji
Liczba modyfikuje kategorie: subst depr num numcol adj ppron12 ppron3 ger pact ppas aglt praet impt pact ppas winien		
sg	Liczba pojedyncza	noga, budynek, stół
pl	Liczba mnoga	nogi, budynki, stoły
Rodzaj modyfikuje kategorie: subst depr num numcol adj ppron12 ppron3 praet pact ppas winien		
m1	męski osobowy	policjant, marynarz,
m2	męski zwierzęcy	pies, baran
m3	męski rzeczowy	parasol, kapelusz, pistolet
f	żeński	portmonetka, broszka
n	nijaki	sprawozdanie, zeznanie, dziecko
Stopień modyfikuje kategorie: adj adv		
Pos	równy	ładny, mądry
Com	wyższy	ładniejszy, mądrzejszy
sup	najwyższy	najładniejszy, najmądrzejszy
Akomodacyjność modyfikuje kategorie: ppron12 ppron3		
congr	uzgadniająca	trzej, siedmioma
rec	rządzająca	trzech, sześciorgiem

Tabela 4.13: Kategorie syntaktyczne rozszerzone o rozpoznawanie liczby, rodzaju, stopnia i akomodacyjności dla języka polskiego przypisane do stanów w modelu Markowa. (Źródło: *opracowane na podstawie (Przepiórkowski i in. 2012)*).

Kategorie przypisane do stanów w ukrytym modelu Markowa rozszerzonym o rozpoznawanie negacji, akcentowości, aglutynacyjności i wokaliczności przedstawia Tabela 4.14

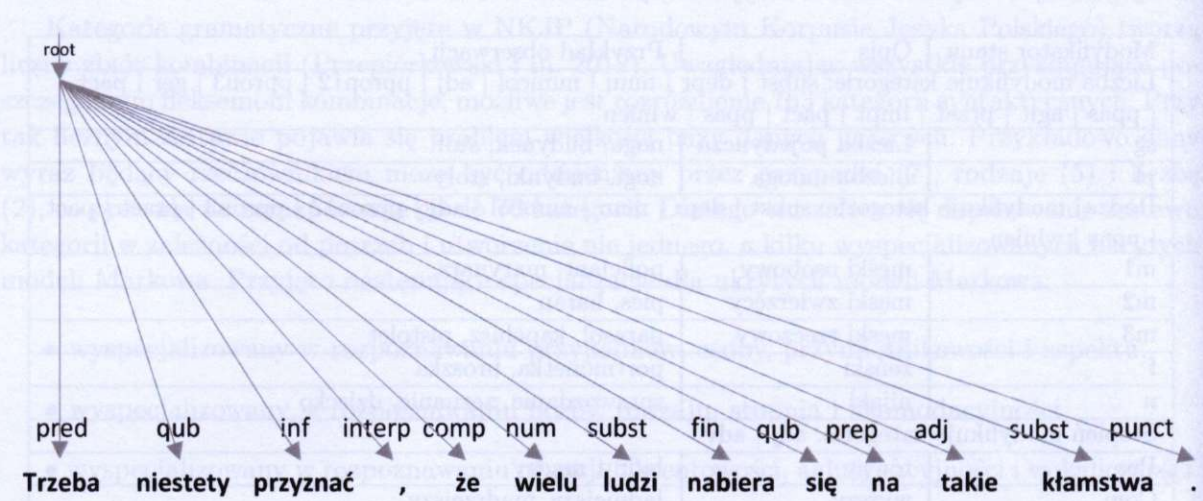
Modyfikator stanu	Opis	Przykład obserwacji
Negacja modyfikuje kategorie: ger pact ppas		
aff	niezanegowana	ruchome, stałe
neg	zanegowana	nieruchome, niestałe
Akcentowość modyfikuje kategorie: ppron12 ppron3		
akc	akcentowana	jego, niego
nakc	nieakcentowana	go, -n
Aglutynacyjność modyfikuje kategorie: praet		
nagl	nieaglutynacyjna	rósł, niósł
agl	aglutynacyjna	rosł-, niosł-
Wokaliczność modyfikuje kategorię: aglt		
wok	wokaliczna	-em, beze
nwok	niewokaliczna	-m, bez

Tabela 4.14: Kategorie syntaktyczne rozszerzone o rozpoznawanie negacji, akcentowości, aglutynacyjności i wokaliczności dla języka polskiego przypisane do stanów w modelu Markowa. (Źródło: opracowane na podstawie (Przepiórkowski i in. 2012)).

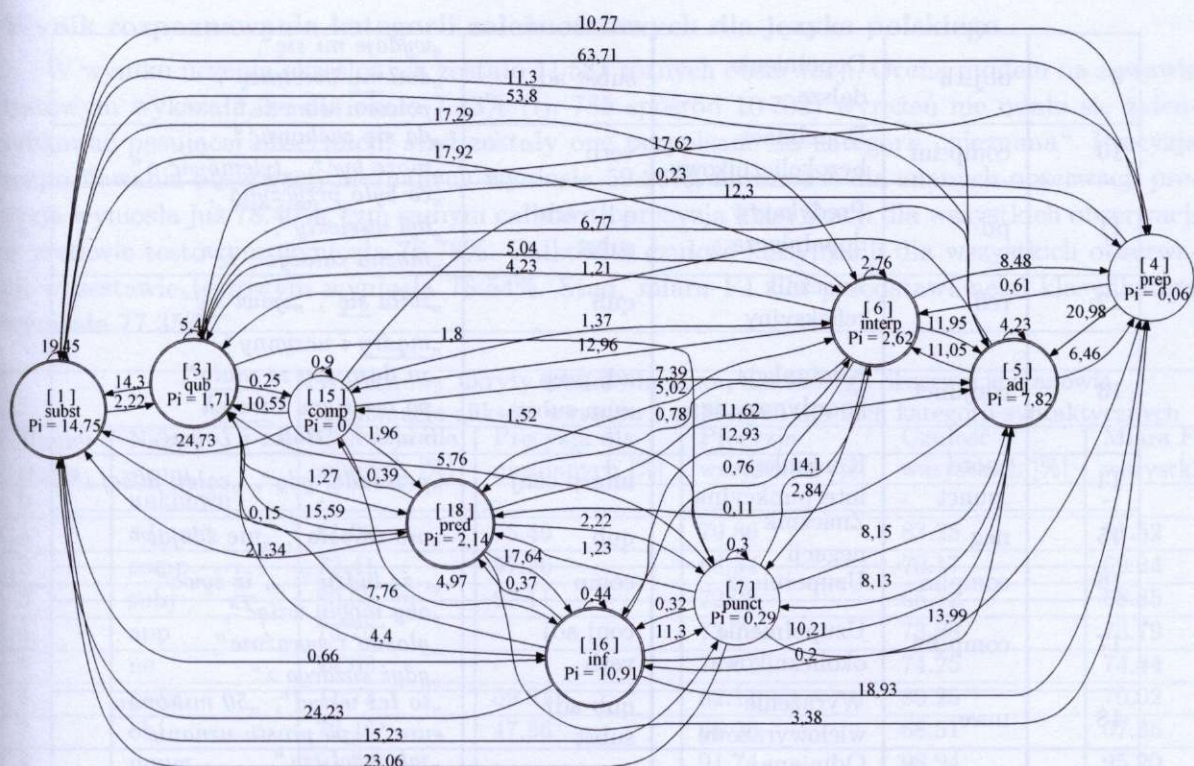
Wybrane przykłady rozpoznawania kategorii syntaktycznych dla języka polskiego

Ze względu na rozbudowaną strukturę modelu Markowa służącego do rozpoznawania kategorii syntaktycznych, jego graficzna prezentacja (patrz Rysunek 4.11) ograniczona została tu jedynie do stanów wykorzystanych w przykładowym zdaniu. Niemniej jednak należy pamiętać, że całkowita przestrzeń stanów jest dużo większa. Przykład zdania w języku polskim:

„Trzeba niestety przyznać, że wielu ludzi nabiera się na takie kłamstwa.”



Rysunek 4.10: Przykład rozpoznawania kategorii syntaktycznych z wykorzystaniem ukrytego modelu Markowa. (Źródło: opracowanie własne).



Rysunek 4.11: Model Markowa prezentujący prawdopodobieństwa przejść między wybranymi kategoriami syntaktycznymi.
(Źródło: opracowanie własne).

4.5.5 Model rozpoznawania kategorii zależnościowych dla języka polskiego

Reguły gramatyki języka polskiego różnią się od tych występujących w języku angielskim. Niemniej jednak pewne analogie w nazewnictwie i stosowanych zasadach pozostają ponadjęzykowe. Jedną z takich zasad jest budowa zdania SVO (ang. *Subject Verb Object*) określająca kolejność wystąpienia podmiotu, orzeczenia i dopełnienia. Kategorie zależnościowe dla języka polskiego przypisane do stanów w ukrytym modelu Markowa przedstawia Tabela 4.15.

Numer stanu	Nazwa stanu	Opis	Część mowy	Przykłady językowe
0	unknown	Wyrażenie nieznane		
1	adjunct	Przydawka	prep conj adv adj qub subst num	„kolo“, „jak“, „jednak“, „lipca“, „już“, „100“
2	comp	Uzupełnienie	prep num subst	„tytułu“, „teren“, „w“, „z“, „na moim biurku“, „do włosów“
3	subj	Podmiot	adj num subst	„jej nikt nie podskoczy“, „każdy z synów założył rodzinę“, „choć <u>wszystko</u> było“, „siedem płócien“
4	app	Przyłożenie	subst	„jego twarz, pan bóg“
5	ne	Skrót	brev	„r. .“, „godz. .“, „prof. .“
6	pred	Predykat	verb adj subst	„robi groźne“, „został podany“, „ten pokój“
7	obj	Dopełnienie bliższe	conj num subst	„zmienia nazwisko“, „planów <u>czy</u> idei“, „które <u>cię</u> cieszą“, „możesz mieć <u>więcej</u> “
8	punct	Znak koniec zdania	punct interp	„ . “

9	obj_th	Dopełnienie dalsze	subst adj	„wydaje <u>mi się</u> “, „dać <u>mu</u> obietnicę“, „powiem <u>wam że</u> “
10	comp_inf	Dopełniacz bezokolicznikowy	verb	„da się zachować“, „może być“, „ <u>tyłemówić</u> o“
11	pd	Predykatyw dopełniacza	adj verb subst	„co było <u>przyczyną</u> “, „był <u>wściekły</u> “, „można założyć, że“
12	refl	Znacznik refleksyjny	qub	„zbiła <u>się</u> “, „zjawi <u>się</u> “
13	conjunct	Koniunkcja koordynowana	adj prep num subst	„ <u>mądry i uczynny</u> “, „za <u>dużo</u> czy za <u>malo</u> “, „ <u>po miesiącu i trzech dniach</u> “, „ <u>Bolka i Lolka</u> “
14	coord_punct	Koniunkcja interpunkcyjna	interp conj	„ <u>przesadę, gdy</u> “, „ <u>całej, dużej i</u> “
15	neg	Znacznik negacji	qub	„ <u>nie</u> zniosła“, „ <u>nie</u> zdając“
16	complm	Uzupełniacz	comp	„ <u>że</u> ludzie“, „ <u>iz</u> żywe“
17	comp_fin	Uzupełnienie okolicznikowe	conj adj verb	„aby <u>mogły wziąć</u> “, „ <u>głośno i wyraźnie</u> “, „ <u>gdyż</u> każdego z“
18	mwe	Wyrażenie wielowyrazowe	qub adj subst	„to <u>też wstyd</u> “, „ <u>50 milionów euro</u> “, „ <u>po prostu</u> uznanie“
19	aglt	Odmiana ruchoma	verb	„jak <u>mogł</u> em“, „ <u>gdyby</u> m wiedział“
20	cond	Enklityka warunkowa	qub	„ <u>stało by się</u> “, „ <u>musiało by nastąpić</u> “
21	abbrev_punct	Znacznik skrótu	punct	„art .“, „prof .“,
22	coord	Koniunkcja koordynująca	conj qub	„ale“, „a“, „ <u>również</u> “, „jako“, „ <u>niż</u> “
23	aux	Pomocniczy	verb	„może <u>zostać</u> “, „ <u>warto było</u> “
24	pre.coord	Pre-koniunkcja	conj	„ <u>albo, to i tak</u> “
25	interp	Znak interpunkcyjny	interp	„ , “
26	ne_		interp subst adj num brev	„ <u>1992 r .</u> “, „ <u>13 czerwca</u> “, „ <u>np .</u> “
27	imp	Znacznik rozkazujący	qub, verb	„ <u>niech</u> pan“, „ <u>mów</u> teraz“

Tabela 4.15: Kategorie zależnościowe dla języka polskiego przypisane do stanów w ukrytym modelu Markowa.

(Źródło: opracowane na podstawie (Wróblewska 2014)).

Wynik rozpoznawania kategorii zależnościowych dla języka polskiego

W wyniku uczenia określonych zostało 11 581 różnych obserwacji. Ocena modelu na zestawie testowym wykazała, że dla około 7.33% (tj. 785 spośród 10 702) wyrażeń nie udało się zidentyfikować pasującej obserwacji, stąd zostały one przypisane do kategorii „nieznana”. Precyzja rozpoznawania obserwacji nieznanych wyniosła 59.26%, natomiast dla znanych obserwacji precyzja wyniosła już 78.40%, tym samym całkowita precyzja klasyfikacji dla wszystkich obserwacji w zestawie testowym wyniosła 76.78%. Całkowita czułość klasyfikacji dla wszystkich obserwacji w zestawie testowym wyniosła 78.54%. Stąd, miara F1 dla przedstawionego klasyfikatora wyniosła 77.35%.

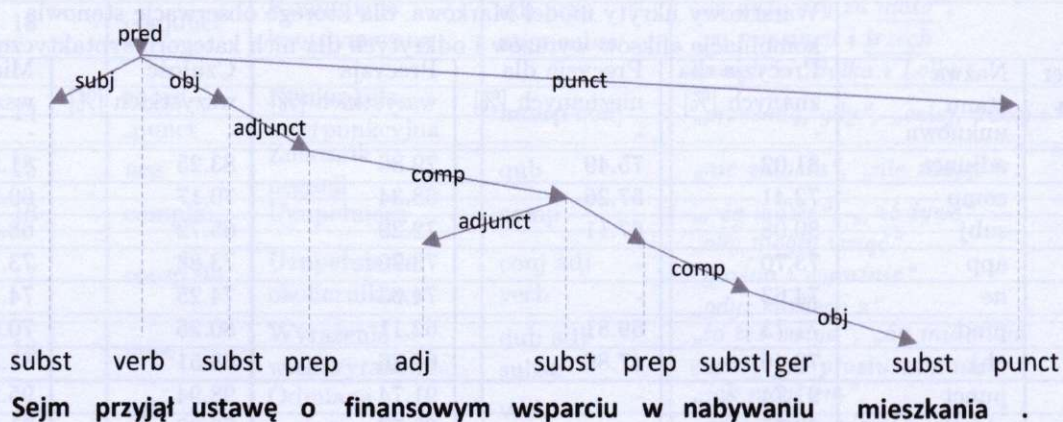
		Warstwowy ukryty model Markowa, dla którego obserwacje stanowią kombinacje afiksów wyrazów i odkrytych dla nich kategorii syntaktycznych				
Numer stanu	Nazwa stanu	Precyzja dla znanych [%]	Precyzja dla nieznanych [%]	Precyzja wszystkich [%]	Czułość wszystkich [%]	Miara F1 wszystkich [%]
0	unknown	-	-	-	-	-
1	adjunct	81.02	75.49	79.86	83.25	81.52
2	comp	72.41	57.26	68.34	70.17	69.24
3	subj	80.08	67.11	72.29	65.72	68.85
4	app	73.70	-	73.70	73.88	73.79
5	ne	74.63	-	74.63	74.25	74.44
6	pred	71.73	59.81	62.11	80.25	70.02
7	obj	72.46	47.86	66.26	68.51	67.36
8	punct	91.74	-	91.74	98.94	95.20
9	obj_th	83.77	-	83.77	79.06	81.34
10	comp_inf	87.60	81.34	86.16	85.03	85.59
11	pd	81.83	-	81.83	82.90	82.36
12	refl	95.53	-	95.53	100	97.71
13	conjunct	81.30	25.92	67.84	61.77	64.66
14	coord_punct	41.50	-	41.50	44.09	42.75
15	neg	95.59	-	95.59	100	97.74
16	complm	81.67	-	81.67	88.87	85.11
17	comp_fin	74.34	-	74.34	86.67	80.03
18	mwe	84.26	-	84.26	67.92	75.21
19	aglt	95.63	-	95.63	100	97.77
20	cond	98.00	-	98.00	100	98.99
21	abbrev_punct	66.19	-	66.19	100	79.65
22	coord	20.37	-	20.37	19.23	19.78
23	aux	80.28	-	80.28	59.74	68.50
24	pre_coord	50.17	-	50.17	50.00	50.08
25	interp	100	-	100	100	100
26	ne_	100	-	100	100	100
27	imp	81.02	-	81.02	80.50	80.75
Razem		78.40	59.26	76.78	78.54	77.35

Tabela 4.16: Dokładność rozpoznawania kategorii zależnościowej z wykorzystaniem ukrytego modelu Markowa dedykowanego dla języka polskiego.
(Źródło: opracowanie własne).

Wybrane przykłady rozpoznawania kategorii zależnościowej dla języka polskiego

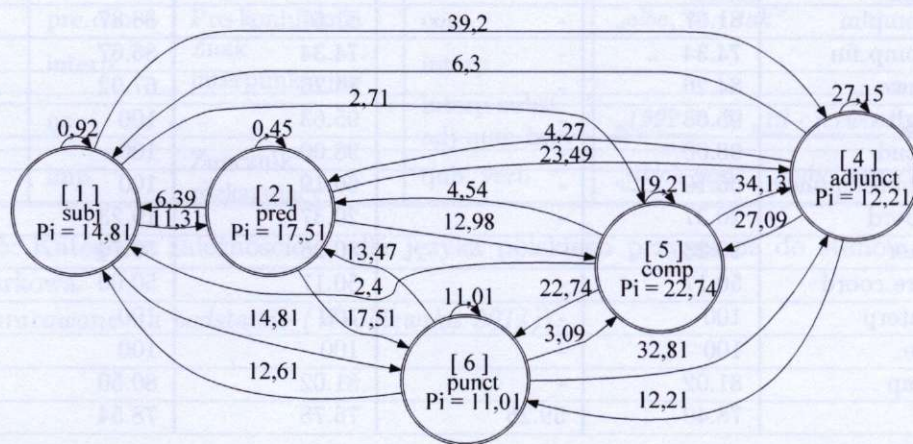
Ze względu na rozbudowaną strukturę modelu Markowa służącego do rozpoznawania kategorii zależnościowej, jego graficzna prezentacja (patrz Rysunek 4.13 i 4.15) ograniczona została tu jedynie do stanów wykorzystanych w przykładowym zdaniu. Niemniej jednak należy pamiętać, że całkowita przestrzeń stanów jest dużo większa. Przykład zdania w języku polskim:

„Sejm przyjął ustawę o finansowym wsparciu w nabywaniu mieszkania.”



Rysunek 4.12: Przykład rozpoznawania części mowy i części zdania w wykorzystaniu ukrytych modeli Markowa.

(Źródło: opracowanie własne).

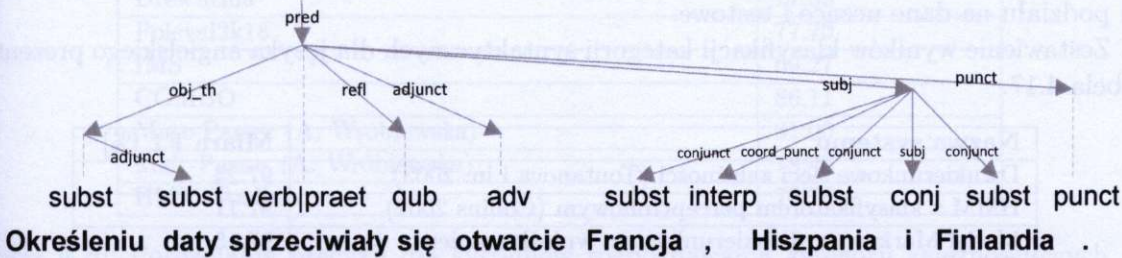


Rysunek 4.13: Model Markowa prezentujący prawdopodobieństwa przejść między wybranymi kategoriami zależnościowymi.

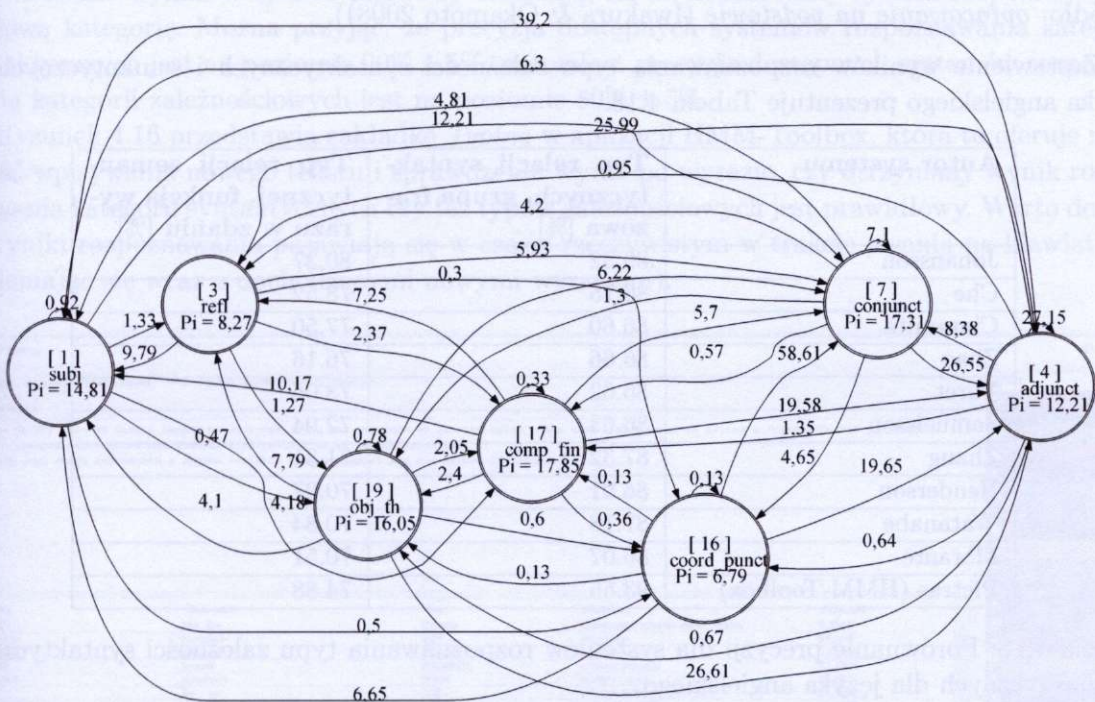
(Źródło: opracowanie własne).

Przykład zdania w języku polskim:

„Określeniu daty sprzeciwiały się otwarcie Francja, Hiszpania i Finlandia.”



Rysunek 4.14: Przykład rozpoznawania kategorii zależnościowych z wykorzystaniem warstwowych ukrytych modeli Markowa.
(Źródło: opracowanie własne).



Rysunek 4.15: Model Markowa prezentujący prawdopodobieństwa przejść między wybranymi kategoriami zależnościowymi.
(Źródło: opracowanie własne).

4.5.6 Podsumowanie i porównanie wyników

Do oceny zadania rozpoznawania danych tekstowych przez zaprezentowane ukryte modele Markowa zastosowano miarę F1 będącą średnią harmoniczną miar precyzji i czułości. Dzięki temu możliwe jest porównanie wyników na tle innych klasyfikatorów o identycznej przestrzeni klas. Wyniki otrzymywane przez prezentowane ukryte modele Markowa można porównać:

- dla języka angielskiego z wynikami prezentowanymi w (Surdeanu i in. 2008, Hajic i in. 2009),
- dla języka polskiego z wynikami prezentowanymi w (Wróblewska 2014, 2018).

Podczas interpretacji wyników należy nadmienić, że z uwagi na trudności związane z dotarciem w poszczególnych źródłach do informacji, które dane zostały (w tamtych systemach) użyte do uczenia, a które do testowania, bardzo ściśle porównanie wyników nie jest możliwe. Niemniej ze względu na ogólnie duży rozmiar dostępnych danych, można oczekiwać otrzymywania w przybliżeniu stałych wyników (precyzji i czułości) niezależnie od przepermutowania danych i ich podziału na dane uczące i testowe.

Zestawienie wyników klasyfikacji kategorii syntaktycznych dla języka angielskiego prezentuje Tabela 4.17.

Nazwa systemu	Miara F1 [%]
Dwukierunkowe sieci zależności (Toutanova i in. 2003)	97.24
HMM z klasyfikatorem perceptronowym (Collins 2002)	97.11
Model Markowa z dwukierunkowym wnioskowaniem (Tsuruoka & Tsujii 2005)	97.15
Maszyna wektorów nośnych (Gimenez & Marquez 2003)	97.05
HMM-Toolbox	95.22

Tabela 4.17: Porównanie miar F1 dla systemów rozpoznawania kategorii syntaktycznych dla języka angielskiego.

(Źródło: *opracowanie na podstawie* (Iwakura & Okamoto 2008)).

Zestawienie wyników rozpoznawania typu zależności syntaktycznych i semantycznych dla języka angielskiego prezentuje Tabela 4.18.

Autor systemu	Typ relacji syntaktycznych, grupa frazowa [%]	Typ relacji semantycznej, funkcja wyrazu w zdaniu [%]
Johansson	89.32	80.37
Che	86.75	78.52
Ciaramita	86.60	77.50
Zhao	86.66	76.16
Yuret	86.62	73.06
Samuelsson	86.63	72.94
Zhang	87.32	71.31
Henderson	86.91	70.97
Watanabe	87.18	70.84
Morante	86.07	70.51
Pietras (HMM-Toolbox)	93.55	74.88

Tabela 4.18: Porównanie precyzji dla systemów rozpoznawania typu zależności syntaktycznych i semantycznych dla języka angielskiego.

(Źródło: *opracowane na podstawie* (Surdeanu i in. 2008)).

Zestawienie wyników klasyfikacji kategorii syntaktycznych dla języka polskiego prezentuje Tabela 4.19.

Nazwa systemu	Precyzja[%]
OpenNLP (Kobyliński & Kieraś 2016)	87.24
Pantera (Acedański 2010)	88.95
WCRFT (Radziszewski 2013)	90.76
Concraft (Waszczuk 2012)	91.07
PoliTa (Kobyliński 2014)	92.01
Toygger (Krasnowska-Kieraś 2017)	92.01
HMM-Toolbox	92.47

Tabela 4.19: Porównanie precyzji dla systemów rozpoznawania kategorii syntaktycznych dla języka polskiego.

(Źródło: *opracowanie własne*).

Zestawienie wyników klasyfikacji typów relacji zależnościowych dla języka polskiego prezentuje Tabela 4.20.

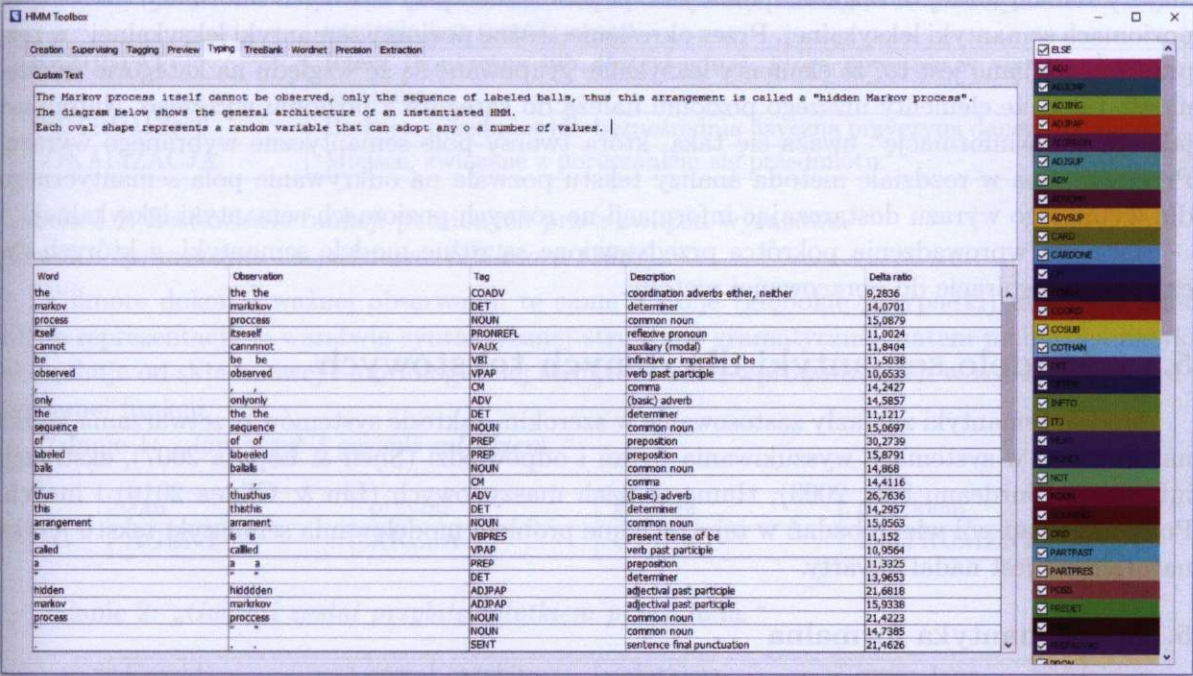
Nazwa systemu	Precyzja LAS [%]
Drewutnia	27.39
Poleval2k18	77.70
IMS	69.27
COMBO	86.11
Mate-Parser (A. Wróblewska)	81.00
Malt-Parser (A. Wróblewska)	78.40
HMM-Toolbox	76.78

Tabela 4.20: Porównanie precyzji dla systemów rozpoznawania kategorii zależnościowych dla języka polskiego.

(Źródło: opracowane na podstawie (Wróblewska 2014, 2018)).

Podsumowując wyniki należy podkreślić, że pewne różnice w przygotowaniu klasyfikatorów jak i danych uczących oraz testowych mają wpływ na jednoznaczność wyników. Niemniej jednak prezentowane wyniki w sposób klarowny określają, na ile dobrze dany system rozpoznaje prawidłową kategorię. Można przyjąć, że precyzja dostępnych systemów rozpoznawania kategorii syntaktycznych jest na poziomie $90\% \pm 5\%$, natomiast precyzja dostępnych systemów rozpoznawania kategorii zależnościowych jest na poziomie $80\% \pm 5\%$.

Rysunek 4.16 przedstawia zakładkę *Typing* w aplikacji HMM-Toolbox, która to oferuje możliwość wpisywania nowego tekstu i sprawdzenia wyraz po wyrazie, czy otrzymany wynik rozpoznawania kategorii syntaktycznych czy też typów zależnościowych jest prawidłowy. Warto dodać, że wyniki rozpoznawania pojawiają się w czasie rzeczywistym w trakcie pisania na klawiaturze (zmieniając się wraz z dochodzącymi nowymi wyrazami).



Rysunek 4.16: Zakładka wpisywania i rozpoznawania nowego tekstu w HMM-Toolbox. (Źródło: opracowanie własne).

Rozdział 5

Opis informacji na różnych poziomach semantyki leksykalnej

Komputerowa analiza semantyczna odnosi się do pewnego abstrakcyjnego zrozumienia znaczenia tekstu. Istnieje wiele interpretacji pojęcia „znaczenie tekstu”. Warto podkreślić, że formalna kategoryzacja uzyskana w wyniku analizy semantycznej różni się od struktur składniowych powierzchniowych i morfologicznych o tyle, że wyraźnie reprezentuje wiedzę na temat obiektów i relacji między nimi. Obiekty te i połączenia często wykraczają poza kategorie językowe i mogą odzwierciedlać zarówno obiekty i relacje w świecie rzeczywistym, jak i abstrakcyjne pojęcia. Dla komputerowej analizy semantycznej, oprócz znajomości gramatyki języka naturalnego, wymagana jest znajomość wielu innych rodzajów wiedzy, w tym pozajęzykowych (znaczeń tekstu, wartości związanych ze strukturami gramatycznymi, znajomość struktury dyskursu, znajomość przedmiotu dyskursu, wiedzy o świecie), a także metod argumentacji (Charwat i in. 2015).

Niniejszy rozdział dotyczy modeli semantyk tekstowych, jak również nowoczesnych metod analizy semantycznej. Uwaga skupiona jest na problemie opisu istotnych informacji na różnych poziomach semantyki leksykalnej. Przez określenie „różne poziomy semantyki leksykalnej” w rozprawie rozumiane jest to, że elementy leksykalne grupowane są ze względu na kategorie znaczeniowe, takie, że elementy niższego poziomu należą do kategorii z poziomu wyższego. Natomiast jako „istotną informację” uważa się taką, która tworzy pole semantyczne wybranego wyrazu. Przedstawiona w rozdziale metoda analizy tekstu pozwala na odkrywanie pola semantycznego dla wybranego wyrazu dostarczając informacji na różnych poziomach semantyki leksykalnej.

Tytułem wprowadzenia pokrótce przedstawione są różne modele semantyki, z których zaczerpnięto inspiracje do opracowanej metody.

5.1 Modele semantyki dla danych tekstowych

Modele semantyki znalazły zastosowanie w szerokim zakresie systemów przetwarzania języka naturalnego: w systemach wyszukiwania pytań i odpowiedzi (Shen & Lapata 2007), ekstrakcji informacji (Surdeanu i in. 2003), tłumaczeniach maszynowych (Liu & Gildea 2010) i innych. Pomimo ponad pół wieku badań w tej dziedzinie problem modelowania semantyki tekstu języka naturalnego jest nadal otwarty.

5.1.1 Semantyka formalna

Powszechnie znanym sposobem zastosowania semantyki wyrażen w modelowaniu języka naturalnego jest jego reprezentacja w języku logiki. Richard Montague wykazał, że język angielski można przełożyć na język formalny (Montague 1970). Sugeruje on, że na poziomie teoretycznym języki formalne i fizyczne nie różnią się istotnie. Semantyka Montague’a oparta jest na logice predykatów – obejmuje rachunek lambda, zaawansowaną strukturę rodzajów i typów funkcji. Prace Montague’a nadały kierunek rozwojowi metod modelowania językiem formalnym aspektów języka naturalnego (tj. charakterystyka obiektów, idiomów, metafor i innych wyrażen). Do

tej pory jednak z powodu niejednoznaczności i elastyczności języka naturalnego zastosowanie formalnego aparatu semantyki w praktyce jest problematyczne ze względu na swoją złożoność.

5.1.2 Funkcjonalny model semantyki

Za jeden z fundamentalnych modeli semantyki tekstu w języku naturalnym uważa się za model oparty na strukturze ról w zdaniu. Nowoczesna formuła tego modelu została zaproponowana w pracach Charlesa Fillmore’a (Fillmore 1968). Wprowadził on pojęcie przypadków semantycznych (nazywanych głębokimi) oraz pierwotność przypadkową (uniwersalia). Uczestnik zdarzenia wykonuje funkcję (tematyczną) wskazaną w tekście przez czasownik. Przypadki posiadają odpowiednie realizacje składniowe i opisują semantyczną warstwę zdania:

- opisują „co uczestnik jest w stanie zrobić z wydarzeniami zachodzącymi wokół niego“;
- opisują „jak ktoś coś zrobił“;
- opisują „komu co się stało“;
- opisują „zmianę, której ktoś został poddany“.

Wiele wprowadzonych pojęć miało umożliwić przedstawienie w formie streszczenia wszystkich możliwych znaczeń sytuacji. Pomimo pewnych rozbieżności istnieje szereg funkcji, które są często stosowane w literaturze (Fillmore 1968).

Nazwa funkcji	Opis
AKCJA	Czynność dokonywana przez agenta.
AGENT	Sprawca danej czynności (powinien być obiektem ożywionym – osobą, zwierzęciem i działać w sposób świadomy).
KO-AGENT	Siła lub opór (przeszkoda), przeciwko któremu akcja jest wykonywana.
OBIEKT	Rzecz (jednostka), która porusza się, zmienia, lub której położenie czy istnienie jest rozpatrywane.
REZULTAT	Rzecz (konkretna lub abstrakcyjna), która powstaje w wyniku danego działania.
INSTRUMENT	Bodziec, przedmiot lub bezpośrednia fizyczna przyczyna danego działania.
LOKALIZACJA	Miejsce, związane z poruszaniem się przedmiotu.

Tabela 5.1: Zestawienie funkcji pełnionych przez związki wyrazowe.
(Źródło: opracowanie na podstawie (Fillmore 1968)).

Fillmore dokonał ważnej obserwacji: te same funkcje (głębokie przypadki) mogą posiadać różne reprezentacje w warstwie syntaktycznej struktury gramatycznej, zatem model semantyki abstrahuje od składniowej i morfologicznej struktury zdań wprowadzonych w korespondencji do tej samej funkcji.

Zdanie 1: „Jan przybił gwóźdź młotkiem.“

Jan	przybił	gwóźdź	młotkiem
AGENT	AKCJA	OBIEKT	INSTRUMENT

Zdanie 2: „Gwóźdź został przybity młotkiem przez Jana.“

Gwóźdź	został przybity	młotkiem	przez Jana
OBIEKT	AKCJA	INSTRUMENT	AGENT

5.1.3 Leksykalny model semantyki

Znaczenie leksykalne i relacje związków wyrazowych są kluczowe z punktu widzenia reprezentacji wiedzy. Do rozwoju tej dziedziny niewątpliwie przyczyniły się prace Pustejovsky’ego (Pustejovsky 1995). Elementy leksykalne grupowane są hierarchicznie ze względu na przynależ-

ność do danej kategorii znaczeniowej, a relacje między nimi są określone i z góry zdefiniowane. Relacje te umożliwiają przejście z jednego elementu leksykalnego do kolejnego. W rozprawie zasadniczo wyróżnia się dwa typy przejść: w poziomie lub w pionie. *Poziome przejście* znaczeniowe elementu leksykalnego określa się mianem synonimizacji, gdy osiągniany element leksykalny jest o tym samym (lub zbliżonym) znaczeniu, lub antonimizacji, gdy osiągniany element leksykalny jest o znaczeniu przeciwnym. *Pionowe przejście* znaczeniowe obejmuje relacje pomiędzy określeniami o bardziej ogólnym znaczeniu z określeniami o znaczeniu bardziej konkretnym.

Leksykalny model semantyki wyróżnia następujące relacje znaczeniowe:

- Rzeczowniki:

- **Hiperonimia:** *Y* jest hiperonimem *X*, jeśli każdy *X* należy do klasy *Y* (*pies* jest hiperonimem dla *jamnik*).
- **Hiponimia:** *Y* jest hiponimem *X*, jeśli każdy *Y* należy do klasy *X* (*jamnik* jest hiponimem dla *pies*).
- **Kohiponimia:** *Y* jest kohiponimem *X*, jeśli *Y* i *X* należą do tej samej klasy *Z* (*jamnik* i *pudel* należą do klasy *pies*).
- **Meronimia:** *Y* jest meronimem *X*, jeśli *Y* jest częścią *X* (*okno* jest meronimem dla *budynek*).
- **Holonimia:** *Y* jest holonimem *X*, jeśli *X* zawiera *Y* (*budynek* jest holonimem dla *okno*).

- Czasowniki:

- **Hiperonimia:** *Y* jest hiperonimem *X*, jeśli działalność *X* zawiera się (w jakiś sposób) w działalności klasy *Y* (*poruszać się* jest hiperonimem dla *jechać*).
- **Troponimia:** *Y* jest troponimem *X*, jeśli działalność *Y* wykonuje (w jakiś sposób) działalność klasy *X* (*seplenić* jest troponimem dla *rozmawiać*).
- **Entailment:** *Y* jest logiczną konsekwencją *X*, jeśli wykonując *X* należy robić *Y* (*spać* jest logiczną konsekwencją dla *chrapać*).
- **Kohiponimia:** *Y* jest kohiponimem *X*, jeśli *Y* i *X* należą do tej samej klasy *Z* (*seplenić* i *szeptać* należą do klasy *mówić*).

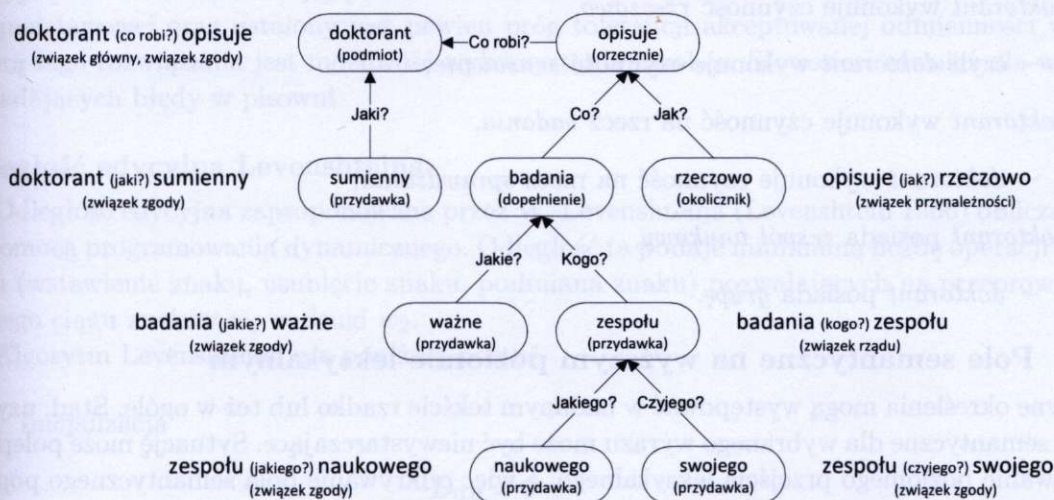
5.2 Odkrywanie pola semantycznego

W prezentowanym rozwiązaniu do opisu istotnych informacji, czyli takich, które tworzą pole semantyczne wybranego wyrazu zastosowano zarówno funkcjonalny, jak i leksykalny model semantyki. Odkrywanie pola semantycznego dla wybranego wyrazu w analizowanym tekście wymaga przeprowadzenia wstępnego rozpoznania kategorii syntaktycznych wyrazów oraz rozpoznania typów relacji między wyrazami. W tym celu zastosowane są ukryte modele Markowa prezentowane w Rozdziale 4.4.

Rozpoznane typy relacji w tekście wykorzystano przy wyborze wyrazów tworzących pole semantycznego wybranego leksemu. Badając pole semantyczne wybranego wyrazu konieczna jest informacja, jaką funkcję pełni ten wyraz w zdaniu, jak również, jakie funkcje pełnią inne wyrazy z nim powiązane. Na podstawie rozpoznanych typów zależności określić można, który z wyrazów dostarcza opisu cech, a który opisu możliwych czynności.

Rozpatrzmy Rysunek 5.1, który ukazuje relacje pomiędzy wyrazami w zdaniu.

„Sumienny doktorant opisuje rzeczowo ważne badania swojego zespołu naukowego.”



Rysunek 5.1: Graficzna reprezentacja relacji między wyrazami w zdaniu.

(Źródło: opracowanie własne).

Pole semantyczne dla wyrazu *doktorant* powstałe na podstawie przedstawionego zdania zawiera cechy i czynności będące bezpośrednio w relacji z wyrazem *doktorant*:

- *doktorant* jest *sumienny*,
 - *doktorant* wykonuje czynność *opisywania*,
- jak i cechy będące w relacji pośredniej:
- *doktorant* wykonuje czynność *rzeczowo*,
 - *doktorant* wykonuje czynność względem *badania*,
 - *doktorant* posiada *zespół naukowy*.

Kategorie syntaktyczne zostaną wykorzystane przede wszystkim przy mapowaniu wyrazów do sieci semantycznej Wordnet. Za jej pomocą możliwe jest dokonanie przejścia pionowego lub poziomego dla elementu leksykalnego (hiperonimia- znaczenie bardziej ogólne, hiponimia- znaczenie bardziej konkretne oraz synonimizacja i antonimizacja). Dzięki przejściu elementu leksykalnego możliwe jest uzyskanie bardziej rozbudowanego, wielopoziomowego opisu istotnych informacji.

Rozpatrując dalej pole semantyczne dla wyrazu *doktorant* z uwzględnieniem przejść leksykalnych dla wyrazów powiązanych (np. przechodząc aż do najwyższego hiperonimu) możemy otrzymywać następujące informacje:

- *doktorant* jest *sumienny*,
 - czyli *doktorant* jest *obowiązkowy*,
 - czyli *doktorant* jest *porządkny*,
 - czyli *doktorant* jest *dobry*;
- *doktorant* wykonuje czynność *opisywania*,
 - czyli *doktorant* wykonuje czynność *raportowania*,
 - czyli *doktorant* wykonuje czynność *informowania*,

- czyli *doktorant* wykonuje czynność *komunikowania*;
- *doktorant* wykonuje czynność *rzeczowo*,
 - czyli *doktorant* wykonuje czynność *sensownie*;
- *doktorant* wykonuje czynność na rzecz *badania*,
 - *doktorant* wykonuje czynność na rzecz *sprawdzania*;
- *doktorant* posiada *zespół naukowy*,
 - *doktorant* posiada *grupe*.

5.2.1 Pole semantyczne na wyższym poziomie leksykalnym

Pewne określenia mogą występować w badanym tekście rzadko lub też w ogóle. Stąd, uzyskanie pola semantyczne dla wybranego wyrazu może być niewystarczające. Sytuację może polepszyć zastosowanie poziomego przejścia leksykalnego, a więc odkrywanie pola semantycznego poprzez synonimizację wybranego wyrazu. Aby uzyskać bardziej rozszerzony opis pola semantycznego, należy dokonać pionowego przejścia leksykalnego wybranego wyrazu, a więc dokonać analizy dla wszystkich hiperonimów, kohiponimów i hiponimów. Należy przy tym zaznaczyć, że uzyskane cechy i relacje dopełniają pole semantyczne wybranego wyrazu jako możliwy dodatkowy opis, ale mogą również stać w sprzeczności ze stanem rzeczywistym. Przykładowo kolejne hiperonimy wyrazu *doktorant* to *kandydat*, a następnie *człowiek*. Rozpatrując zdania przytoczone już w tej rozprawie dokonamy odkrywania pola semantycznego dla wybranego wyrazu *człowiek*.

„Biegli sądowi zostali powołani.“

„Chłopcy biegli przez pole.“

„Ignacy chętnie opowiada o swojej rodzinie.“

„Trzeba niestety przyznać, że wielu ludzi nabiera się na takie kłamstwa.“

„Sumienny doktorant opisuje rzeczowo ważne badania swojego zespołu naukowego.“

„Jan przybił gwóźdź młotkiem.“

Zdania te, choć nie są ze sobą powiązane bezpośrednio, to jednak zawierają pewną pragmatyczną wiedzę na temat istoty żywej jaką jest człowiek. Zarówno „Biegli sądowi“, „Chłopcy“, „Ignacy“, „wielu ludzi“, „doktorant“ i „Jan“ na wyższym poziomie semantyki leksykalnej grupowani są pod kategorią *człowiek*. Z przytoczonych zdań wiemy więc, że człowiek może „zostać powołany“, „biegać“, „opowiadać“, „nabierać się“, „opisywać“ czy też „przybijając“. Wiemy również, że człowiek pewne czynności może wykonywać „chętnie“ i może być „sumienny“, może mieć „rodzinę“ ale i „zespół“, można go spotkać na „polu“, a ponadto ma pewne umiejętności posługiwania się „młotkiem“.

Pole semantyczne odkryte dla wyrazu *człowiek* może dopełniać opis wyrazu *doktorant*. Na potrzeby rozróżnienia, pole semantyczne wyrazu uzyskane na podstawie poziomego przejścia leksykalnego nazywać będziemy polem semantycznym bliższym, natomiast pole semantyczne uzyskane na podstawie pionowego przejścia leksykalnego nazywać będziemy polem semantycznym dalszym.

5.2.2 Mapowanie na Wordnet

Dla języków angielskiego i polskiego dostępne są słowniki stanowiące złożoną sieć semantyczną znane, odpowiednio, pod nazwami: Wordnet oraz Słownosieć. Wyrazy zawarte w tych słownikach są podane w formie podstawowej (czasowniki w formie bezokolicznikowej, rzeczowniki w mianowniku w liczbie pojedynczej, przymiotniki w mianowniku w liczbie pojedynczej rodzaju męskiego w stopniu równym) co sprawia, że aby prawidłowo skorzystać z zasobów należy analizowane wyrazy również sprowadzić do formy podstawowej. Ma to szczególne znaczenie zwłaszcza dla języka polskiego, gdzie odmiana wyrazu ma wpływ na końcówkę fleksyjną.

W rozprawie jako prosty zabieg ułatwiający rozpoznanie wyrazu wykorzystano odległość edycyjną Levenshteina. Na jej podstawie określony jest stopień podobieństwa wyrazu do formy podstawowej oraz ustalony jest pewien próg tolerancji akceptowanej odmienności wyrazu. Zaletą tego rozwiązania jest możliwość wykorzystania zasobów Słownosieci również dla wyrazów posiadających błędy w pisowni.

Odległość edycyjna Levenshteina

Odległość edycyjna zaproponowana przez W. Levenshteina (Levenshtein 1966) obliczana jest za pomocą programowania dynamicznego. Odległość ta podaje minimalną liczbę operacji edycyjnych (wstawienie znaku, usunięcie znaku, podmiana znaku) pozwalających na przeprowadzenie jednego ciągu znaków w_1 w drugi w_2 .

Algorytm Levenshteina ma poniższą postać.

1. Inicjalizacja

$$D_{0,0} = 0 \quad (5.1)$$

$$D_{i,0} = i, \quad 1 \leq i \leq |w_1| \quad (5.2)$$

$$D_{0,j} = j, \quad 1 \leq j \leq |w_2| \quad (5.3)$$

2. Indukcja

$$D_{i,j} = \min \begin{cases} D_{i-1,j-1} + 0, & w_1(i) = w_2(j), \\ D_{i-1,j-1} + 1, & w_1(i) \neq w_2(j), \\ D_{i-1,j} + 1, & |w_1| < |w_2|, \\ D_{i,j-1} + 1, & |w_1| > |w_2|. \end{cases} \quad (5.4)$$

3. Zakończenie

$$\text{LevDist} = D_{|w_1|,|w_2|} \quad (5.5)$$

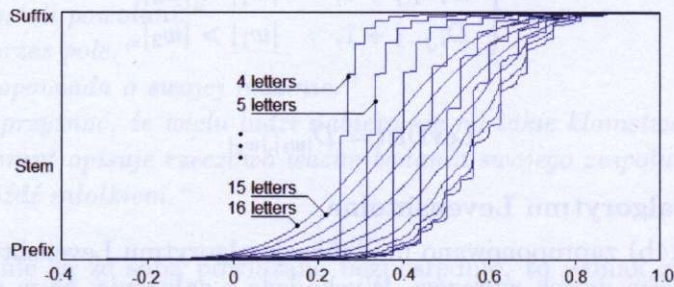
Wagowa modyfikacja algorytmu Levenshteina

W pracy (Pietras 2014b) zaproponowano modyfikację algorytmu Levenshteina pozwalającą na rozmytą ocenę odległości dwóch wyrazów. Wychodząc z założenia, że w gramatyce języka polskiego w wyniku koniugacji i deklinacji zmianie najczęściej ulega sufix wyrazu, stąd wartość kosztu edycji znaku ważona jest współczynnikiem odległości litery od początku wyrazu. Dlatego też, różnice pomiędzy ostatnimi literami badanych wyrazów będą w mniejszym stopniu wpływały na wynikową odległość. Dla uproszczenia pierwsze trzy litery wyrazu traktowane są jako jego stałe (nie podlegające odmianie) i jeżeli w porównywanych wyrazach którakolwiek z tych pierwszych liter okaże się różna, to wyrazy te zostaną sklasyfikowane jako zupełnie odmienne (brak podobieństwa). Jak zauważono w pracy (Weigel & Fein 1994), wagowa modyfikacja algorytmu Levenshteina powoduje, że otrzymywany wynik przestaje być metryką. Natomiast z punktu widzenia aplikacji rozpoznawania polskich wyrazów tego typu modyfikacja pozwala na lepsze dopasowanie znaczenia odmienianych wyrazów.

Dla przykładu porównując wyrazy „rozwiązać” i „rozwinąć” otrzymamy wynik 0.67. Natomiast porównując wyraz „rozwiązać” z wyrazami pochodzącymi z jego koniugacji: „rozwiązali”, „rozwiąż”, „rozwiązałem” otrzymamy odpowiednio: 0.80, 0.78, 0.82. Tabela 5.4. przedstawia wyniki dopasowania łańcuchów liter (odpowiednik wyrazu), gdzie zmianie ulegają litery na różnych pozycjach. Łańcuchem wzorcowym jest „oooooooo” natomiast litera *x* symuluje różniące się znaki.

Ciąg liter	Dopasowanie	Ciąg liter	Dopasowanie
„oooooooooooo”	1.000	„ooooooooxx”	0.789
„ooooooooox”	0.900	„ooooooooxxx”	0.664
„ooooooooxo”	0.889	„ooooooooxxx”	0.521
„ooooooooxoo”	0.875	„ooooooooxox”	0.764
„ooooooooxooo”	0.857	„ooooooooxxox”	0.621
„ooooooooxoooo”	0.833	„ooooooooxxxox”	0.454
„oooooxoooo”	0.800	„ooooooooooooox”	0.909
„oooxoooo”	0.800	„ooooooooooooxx”	0.833
„ooxoooo”	0.000	„oooooooooooo”	0.900
„oxoooo”	0.000	„oooooooooooo”	0.800
„xoooo”	0.000	„ooooooooxxxx”	0.154

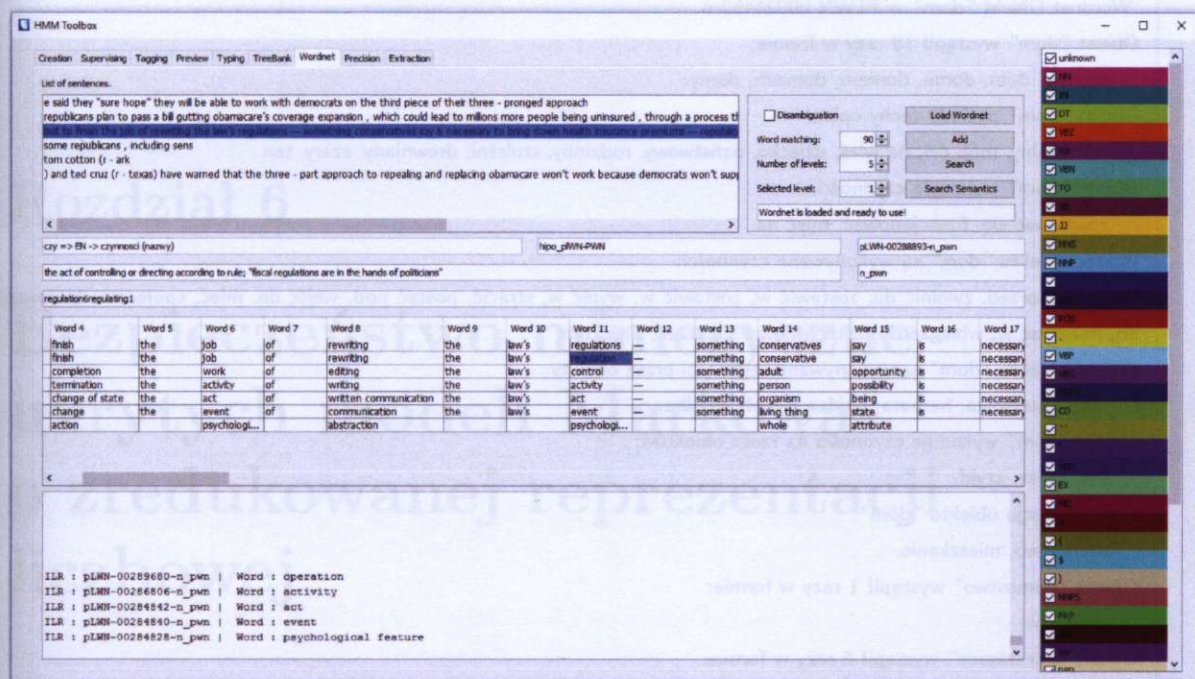
Tabela 5.4: Wartość dopasowania ciągów literowych, gdzie zmianie ulega kombinacja znaków na różnej pozycji.
(Źródło: opracowanie własne).



Rysunek 5.2: Graficzna reprezentacja wartości dopasowania ciągów literowych w zależności od długości ciągu.
(Źródło: opracowanie własne).

5.3 Przykłady użycia

W aplikacji HMM-Toolbox, w zakładce „Wordnet” zaprezentowany jest zestaw funkcji umożliwiających dokonanie opisu istotnych informacji na różnych poziomach semantyki leksykalnej (patrz Rysunek 5.3). W tym celu program korzysta z zasobów sieci semantycznej słownika Wordnet. Wyrazy w analizowanym tekście są oznakowane kategoriami syntaktycznymi oraz typami relacji zależnościowych, aby wyodrębnić takie związki wyrazowe jak podmiot czy orzeczenie. Dla wybranego wyrazu sprawdzane są wystąpienia i powiązania w analizowanym tekście. Pionowe i poziome przejścia elementu leksykalnego następują iteracyjnie (poczynając od wybranego wyrazu oraz przypisanych mu cech i czynności) aż do wyszukania wszystkich istotnych relacji zawartych w sieci semantycznej słownika Wordnet.



Rysunek 5.3: Opis informacji na różnych poziomach semantyki leksykalnej w HMM-Toolbox. (Źródło: opracowanie własne).

Przedstawiona metoda opisu istotnych informacji na różnych poziomach semantyki leksykalnej pozwala na analizę semantyczną wobec wybranego wyrazu na podstawie badanych dokumentów tekstowych. Tym samym tworzony opis cech i czynności przypisanych do wybranego wyrazu zależy od przypadków jego użycia w tekście. Uzyskana jest więc pewna wiedza propozycyjalna na temat wybranego wyrazu.

W celu lepszego zaprezentowania możliwości wykorzystania przedstawionej w tym rozdziale metody opisu informacji na różnych poziomach semantyki leksykalnej przeanalizowany został tekst pochodzący z banku drzew Składnica (część testowa banku drzew, 988 zdań). Zakłada się, że zawarte w nim zdania nie są ze sobą powiązane. Celem analizy jest określenie znaczenia wybranego wyrazu przez zbadanie jego wystąpień i powiązań w danym tekście. Innymi słowy postaramy się uzyskać pewną wiedzę na temat wybranego wyrazu badając jego przypadki użycia w tekście.

Poniżej znajduje się wydruk z aplikacji HMM-Toolbox przedstawiający opis obiektu *dom* na podstawie analizowanego tekstu.

Wordnet Obiekt "dom" = PLWN-00028452-n

Obiekt "dom" wystąpił 18 razy w formie:

domów, dom, domu, domem, domach, domy.

Obiekt "dom" posiada cechy opisu:

mieszkalny, mój, ów, na wieś, dziecko, państwowy, rodzinny, stuletni, drewniany, szary, ten.

Obiekt "dom" wykonuje czynności:

znajdować się, funkcjonować, mieć na.

Wobec obiektu "dom" są wykonywane czynności:

wybiec przed, zwolnić do, zostawić w, wypić w, stracić, postać pod, wejść do, mieć, społecznie budować, w do, mieszkać w, wbiegać do, zatrzymywać się przed, mieć ten.

Wobec obiektu "dom" są wykonywane czynności przez obiekty:

raisa, parasolka, herbata, najder, włoch, minibus.

Obiekt "dom" wykonuje czynności na rzecz obiektów:

kraj, afisz, szyld.

Synonimizacja obiektu "dom":

domostwo, mieszkanie.

Obiekt "domostwo" wystąpił 1 razy w formie:

domostw.

Obiekt "mieszkanie" wystąpił 6 razy w formie:

mieszkaniu, mieszkanie, mieszkania.

Uzyskane pole semantyczne bliższe dla obiektu dom:

Obiekt "dom" posiada cechy przynależności:

swój.

Wobec obiektu "dom" są wykonywane czynności:

porzucać, zapanować w, utracić, przepisać na, znajdować się w.

Wobec obiektu "dom" są wykonywane czynności przez obiekty:

cywile, nastrój, Najder.

Hiperonimia obiektu "dom":

budynek, konstrukcja.

Obiekt "budynek" wystąpił 1 razy w formie:

budynku.

Obiekt "konstrukcja" wystąpił 1 razy w formie:

konstrukcję.

Uzyskane pole semantyczne dalsze dla obiektu dom:

Wobec obiektu "dom" są wykonywane czynności:

zlokalizować w, zniszczyć.

Wobec obiektu "dom" są wykonywane czynności przez obiekty:

woda.

Hiperonimia uzyskanych cech i czynności:

Obiekt "dom" posiada cechy przynależności:

tu, niepełnoletni, człowiek.

Obiekt "dom" posiada cechy opisu:

czyjś, wspólny, stary, roślinny, naturalny, chłodny.

Obiekt "dom" wykonuje czynności:

mieć miejsce, funkcjonować, cechować się.

Wobec obiektu "dom" są wykonywane czynności:

opuścić, wypuścić, pozwolić, spowodować, przybywać, docierać, dostawać się, przemieszczać się, zrobić, zająć się, zdarzyć, wytrzymać, pozostać postać pod, utrzymać się, osiągnąć, dotrzeć, dostać się, zużywać, cechować się, przebywać, przestawać

Wobec obiektu "dom" są wykonywane czynności przez obiekty:

parasol, osłona, ochrona, człowiek, napój, substancja, Europejczyk, bus, samochód, dwuślad, pojazd, dzieło, tablica.

Obiekt "dom" wykonuje czynności na rzecz obiektów:

tu, państwo, organizacja, zespół, grupa, ludzie.

Rozdział 6

Bezpieczeństwo numeryczne ukrytych modeli Markowa o zredukowanej reprezentacji liczbowej

Jak już wspomniano w (Rabiner 1989), obliczenia w ramach HMM mogą stać się kłopotliwe, zwłaszcza w przypadku długich sekwencji. Nawet jeśli obliczenia te są wykonywane w przestrzeni logarytmicznej, należy pamiętać, że istnieje zawsze pewna granica długości badanej sekwencji, poza którą obliczenia mogą stać się numerycznie niestabilne. Zjawisko to należy rozpatrywać w parze z rozmiarem (szerokością bitową) reprezentacji liczb zmiennoprzecinkowych. W celu implementacji FPGA w dalszej części pracy zastosowana zostanie redukcja standardowej reprezentacji o podwójnej precyzji (64 bity) do precyzji: pojedynczej (32 bity), połówkowej (16 bitów) lub ćwiartkowej (8 bitów). Przedstawiony poniżej lemat stanowi główny teoretyczny wynik pracy. Pokazuje on ilościowo, w jaki sposób można ustalić długość najkrótszej sekwencji niebezpiecznej numerycznie w terminach skrajnych prawdopodobieństw występujących w modelu HMM oraz przy pomocy zadanej szerokości bitowej mantysy dla reprezentacji zmiennoprzecinkowej. Wynik ten został opublikowany w pracy (Pietras & Klęsk 2017).

6.1 Lemat „o długości sekwencji niebezpiecznej numerycznie“

Niech $p > 0$ i $P > 0$ oznaczają odpowiednio najmniejsze niezerowe i największe prawdopodobieństwa występujące w modelu HMM (tj. skrajne prawdopodobieństwa z macierzy przejść, emisji lub rozkładu początkowego). Załóżmy, że obliczenia są wykonywane w ramach logarytmowanej wersji algorytmu Viterbiego, a model jest przechowywany w ograniczonej reprezentacji zmiennoprzecinkowej z wykorzystaniem mantysy o szerokości m . Wtedy, najkrótsza sekwencja niebezpieczna numerycznie – tj. taka, która może powodować błędy numeryczne wynikające z ograniczonej reprezentacji nie zaś standardowe błędy zaokrągleń – jest długości

$$\Theta \left(2^{m - (\log_2 |\log p| - \log_2 |\log P|) + 1} \right). \quad (6.1)$$

Dla przypomnienia warto podkreślić, że chodzi tu o dokładny rząd asymptotyczny w odróżnieniu od notacji dużego O (górne ograniczenie asymptotyczne) czy też dużego Ω (dolne ograniczenie asymptotyczne). Innymi słowy, jeżeli w naszym przypadku mówimy, że niebezpieczna sekwencja ma długość $T^* = \Theta(f(m, p, P))$, gdzie f to funkcja w powyższym wzorze, to oznacza to, że istnieją pewne dwie stałe c_1, c_2 takie że dokładna wartość T^* mieści się w widełkach: $c_1 \cdot f(m, p, P) \leq T^* \leq c_2 \cdot f(m, p, P)$.

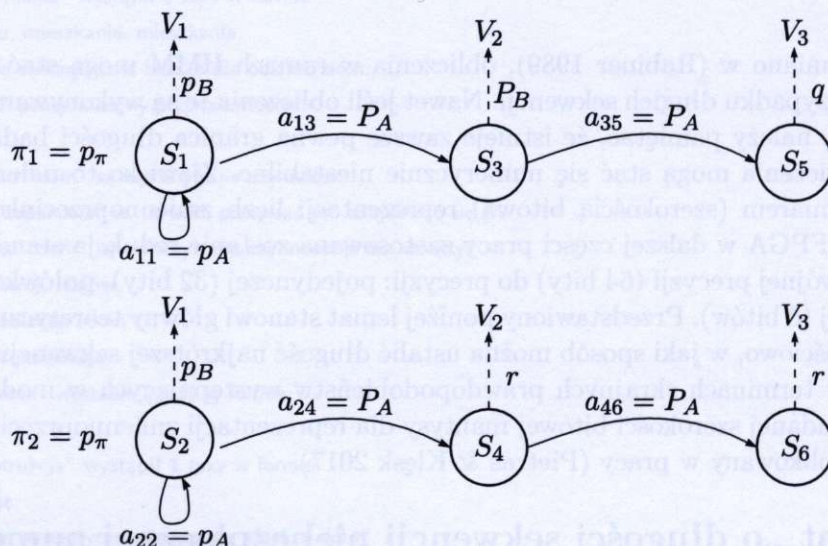
Szkic dowodu: Zaaranżujemy sytuację, w której dla pewnej sekwencji obserwacji możliwe będą tylko dwie ścieżki Viterbiego o bardzo zbliżonych prawdopodobieństwach. Początkowy długi

fragment sekwencji będzie wskazywał na remis pomiędzy ścieżkami, dopiero dwie ostatnie obserwacje przeważą szalę na rzecz jednej ze ścieżek (w przypadku obliczeń dokładnych). Pokażemy, że w przypadku obliczeń numerycznych w ramach logarytmowanej wersji algorytmu Viterbiego możliwe będzie rozstrzygnięcie przeciwne.

Dowód: W notacji małe litery będą wskazywały na małe prawdopodobieństwa (i skojarzone z nimi logarytmy), duże litery będą wskazywały na duże prawdopodobieństwa (i ich logarytmy). Rozważmy minimalne i maksymalne prawdopodobieństwa w macierzach π , A , B (uwaga: w przypadku minimalnych, chodzi o minimalne niezerowe):

$$\begin{aligned} p_\pi &= \min \{ \pi_i : 1 \leq i \leq N, \pi_i > 0 \}, & l_\pi &= \log p_\pi, \\ p_A &= \min \{ a_{ij} : 1 \leq i, j \leq N, a_{ij} > 0 \}, & l_A &= \log p_A, \\ p_B &= \min \{ b_{ik} : 1 \leq i \leq N, 1 \leq k \leq M, b_{ik} > 0 \}, & l_B &= \log p_B, \\ P_A &= \max \{ a_{ij} : 1 \leq i, j \leq N \}, & L_A &= \log P_A, \\ P_B &= \max \{ b_{ik} : 1 \leq i \leq N, 1 \leq k \leq M \}, & L_B &= \log P_B. \end{aligned} \quad (6.2)$$

Rozważmy szczególny model HMM, w którym ważne są tylko stany i emisje przedstawione przez Rysunek 6.1.



Rysunek 6.1: Szczególny model HMM dla lematu.

(Źródło: opracowanie własne).

Należy zaznaczyć, że przejścia pomiędzy stanami S_1 i S_2 są niemożliwe, $a_{12} = a_{21} = 0$; a także że wszystkie inne stany modelu $S_{i>6}$ (pominięte na rysunku) mają zerowe prawdopodobieństwa emisji dla obserwacji: V_1, V_2, V_3 . Oprócz przejść wskazanych przez Rysunek 6.1 stany $S_1, \dots, S_6$ mogą wykonywać przejścia do innych stanów $S_{i>6}$, jednakże te nie będą ważne w przykładzie. Prawdopodobieństwa emisji q, r można traktować jak parametry, które zostaną odpowiednio dobrane dla rozstrzygnięcia remisu. Niech $l_q = \log q$ i $l_r = \log r$. Przypuśćmy, że na wejściu pojawiła się następująca sekwencja obserwacji do zbadania:

$$\underbrace{(V_1, V_1, \dots, V_1, V_2, V_3)}_{T-2}.$$

Łatwo zauważyć, że możliwe są tylko dwie ścieżki stanów, które mogły wyprodukować powyższą sekwencję. Są to:

$$\begin{aligned} (S_1, S_1, \dots, S_1, S_3, S_5), \\ (S_2, S_2, \dots, S_2, S_4, S_6). \end{aligned}$$

Nazwijmy je odpowiednio ścieżką „nieparzystą“ i „parzystą“ (ze względu na indeksy biorące w nich udział). Iloczyn prawdopodobieństw i odpowiadająca mu suma logarytmów dla ścieżki „nieparzystej“ wynoszą:

$$\underbrace{p_\pi \cdot p_B}_{t=1} \cdot \underbrace{p_A \cdot p_B}_{t=2} \cdot \dots \cdot \underbrace{p_A \cdot p_B}_{t=T-2} \cdot \underbrace{P_A \cdot P_B}_{t=T-1} \cdot \underbrace{P_A \cdot q}_{t=T}, \quad (6.3)$$

$$\underbrace{l_\pi + l_B}_{t=1} + \underbrace{l_A + l_B}_{t=2} + \dots + \underbrace{l_A + l_B}_{t=T-2} + \underbrace{L_A + L_B}_{t=T-1} + \underbrace{L_A + l_q}_{t=T}. \quad (6.4)$$

Analogicznie dla ścieżki „parzystej“ mamy:

$$\underbrace{p_\pi \cdot p_B}_{t=1} \cdot \underbrace{p_A \cdot p_B}_{t=2} \cdot \dots \cdot \underbrace{p_A \cdot p_B}_{t=T-2} \cdot \underbrace{P_A \cdot r}_{t=T-1} \cdot \underbrace{P_A \cdot r}_{t=T}, \quad (6.5)$$

$$\underbrace{l_\pi + l_B}_{t=1} + \underbrace{l_A + l_B}_{t=2} + \dots + \underbrace{l_A + l_B}_{t=T-2} + \underbrace{L_A + l_r}_{t=T-1} + \underbrace{L_A + l_r}_{t=T}. \quad (6.6)$$

Przypuśćmy, że w rzeczywistości bardziej prawdopodobna jest ścieżka „parzysta“ (tzn. w przypadku obliczeń dokładnych). Oznacza to, że spełniona jest następująca nierówność (po uproszczeniu wyrazów remisowych i zredukowaniu do samej końcówki):

$$P_B \cdot q < r^2. \quad (6.7)$$

Jednocześnie chcielibyśmy pokazać możliwość błędu numerycznego polegającego na zdarzeniu, że algorytm wskaże jako lepszą ścieżkę „nieparzystą“. Błąd taki może wystąpić wtedy, gdy w chwili $t = T - 1$ dodanie wyrazu L_B (bliskiego zeru ponieważ P_B jest maksymalne) do bieżącej sumy, która jest odpowiednio dużego rzędu, nie zmieni wyniku. A zatem chcemy spowodować przeciwny kierunek nierówności w (6.7) zastępując równoważnie P_B jedynką ze względu na nierozróżnialność wyniku, tj.:

$$1 \cdot q > r^2, \quad (6.8)$$

co, patrząc na sumę logarytmów, odpowiada:

$$\log 1 + \log q > 2 \log r. \quad (6.9)$$

Zatem, aby spełnić jednocześnie (6.7) i (6.8) należy dobrać q jako:

$$r^2 < q < r^2 / P_B. \quad (6.10)$$

Dla uproszczenia rozważań numerycznych zaniedbajmy teraz rozróżnienie pomiędzy prawdopodobieństwami pochodzącymi z A , B lub π ; tj. utożsamimy $p_A = p_B = p_\pi = p$ oraz $P_A = P_B = P_\pi = P$. Tym samym niech $l_A = l_B = l_\pi = l$ oraz $L_A = L_B = L_\pi = L$. Innymi słowy interesują nas tylko małe i duże wyrazy (co do rzędu wielkości) w sumie logarytmów. I tak suma (6.4) dla ścieżki „nieparzystej“ upraszcza się do

$$\underbrace{l + l}_{t=1} + \underbrace{l + l}_{t=2} + \dots + \underbrace{l + l}_{t=T-2} + \underbrace{L + L}_{t=T-1} + \underbrace{L + l_q}_{t=T}, \quad (6.11)$$

a suma (6.6) dla ścieżki „parzystej“ do

$$\underbrace{l + l}_{t=1} + \underbrace{l + l}_{t=2} + \dots + \underbrace{l + l}_{t=T-2} + \underbrace{L + L}_{t=T-1} + \underbrace{L + l_r}_{t=T}. \quad (6.12)$$

Przed krytycznym momentem $t = T - 1$ bieżąca wartość obu powyższych sum powstaje poprzez dodanie $2(T - 2)$ logarytmów z małych prawdopodobieństw, czyli wartości bezwzględnie

dużych; łatwo wymusić $|l| \gg |L|$. A zatem staramy się w szybkim tempie „wyczerpać” m cyfr znaczących (w bazie dwójkowej) mantysy tuż przed dodaniem składnika L , który jest małego rzędu. Wielkość $\log_2 |L|$ można traktować właśnie jako *rzęd wielkości* L w bazie dwójkowej. A zatem wspomnianą sytuację błędu, gdy następuje brak rozróżnienia wyniku po dodaniu L , można opisać nierównością:

$$\frac{2^{m-|\log_2 |L||+1}}{2(T-2)|l|} \leq 1. \quad (6.13)$$

Licznik ułamka reprezentuje rząd wielkości maksymalnej możliwej do zapisania liczby na mantysie szerokości m po uwzględnieniu odjęcia $|\log_2 |L||$ cyfr znaczących krytycznego składnika. Mianownik reprezentuje tempo, w jakim wyczerpujemy ten dostępny rząd wielkości, poprzez $2(T-2)$ -krotne dodawanie l w początkowej fazie obliczeń. Rozwiązując (6.13) ze względu na T (długość sekwencji) otrzymujemy:

$$\begin{aligned} T &\geq 2 + \frac{2^{m-|\log_2 |L||+1}}{2|l|} \\ &= 2 + 2^{m+\log_2 |L|+1} \cdot 2^{\log_2 |2l|^{-1}} \\ &= 2 + 2^{m+\log_2 |L|-\log_2 |2l|+1} \\ &= 2 + 2^{m-(\log_2 |2l|-\log_2 |L|)+1} = T^*. \end{aligned} \quad (6.14)$$

Póki co pokazane zostało, że mogą istnieć sekwencje niebezpieczne numerycznie o długości większej od lub równej pewnej wartości progowej, tj. przypadek: $T \geq T^*$. Biorąc pod uwagę sens notacji dużego Θ w lemacie, aby dokończyć dowód, należy jeszcze pokazać przypadek $T \leq T^*$ – ograniczając tym samym z góry najkrótszą spośród takich sekwencji. Tu wystarczy zauważyć, że jeżeli w naszym przykładzie w początkowym fragmencie ścieżki weźmiemy pod uwagę prawdopodobieństwa większego rzędu niż minimalne p , to dodawane logarytmy prawdopodobieństw nie wyczerpią tak szybko dostępnego zakresu. Tym samym sytuacja błędu nierozróżnialnego wyniku może co najwyżej wystąpić później. A zatem *najkrótsza* niebezpieczna sekwencja jest długości $\Theta(T^*)$. ■

Uwaga (1).

Ze wzoru na T^* (6.14) widać, że ważna jest różnica pomiędzy rzędami wielkości logarytmów (tj. $\log_2 |2l| - \log_2 |L|$) ze skrajnych prawdopodobieństw w modelu HMM. Im ta różnica jest większa, tym mniej pozostaje swobodnych cyfr znaczących w mantysie.

Uwaga (2).

O ile zewnętrzny logarytm oddający rząd wielkości musi być dwójkowy – $\log_2 \dots$ – tak wewnętrzne logarytmy schowane w $l = \log p$ i $L = \log P$ mogą być wyrażone w dowolnej podstawie.

6.2 Przykłady numeryczne dla lematu

Poniżej przedstawiono Listing 6.1 przykładowej procedury w Javie naśladującej eksperyment z lematu. Procedura testuje typ `float` (czyli $m = 23$). Stałe p, P, q, r można dobierać dowolnie zgodnie z założeniami lematu. W tej chwili nastawione są wartości, tak aby moment T^* oddany przez program był dokładny, zgodny ze wzorem (6.14). Nastawy odchodzące od potęg dwójki spowodują, że program da wynik co najwyżej bardziej optymistyczny, tj. że niebezpieczna sekwencja będzie dłuższa niż wg (6.14).

Listing 6.1: Procedura dokonująca eksperyment zgodnie z lematem.

```

1 public static long testViterbiFloats(long maxT) {
2     log("TEST VITERBI (FLOATS) FOR T: " + ((maxT == Long.MAX_VALUE) ? "INFINITY" : (maxT + 2)));
3
4     //minimum non-zero probability p and its logarithm (p = p-\pi = p-A = p-B)
5     final float p = (float) Math.exp(-16.0);
6     final float l = (float) Math.log(p);
7
8     //maximum probability P and its logarithm
9     //(we start by entering logarithm first, so that its value can be a power of 2:
10    // this allows later to obtain accurate length of non-safe sequence)
11    final float L = (float) (-1.0 * Math.pow(2.0, -10.0));
12    final float P = (float) Math.exp(L);
13
14    //auxiliary constants
15    final float r = (float) Math.sqrt(2 * p);
16    final float lr = (float) Math.log(r);
17
18    // r^2 < q < r^2 / P
19    final double epsilon = 0.5 * (1.0 / P - 1.0);
20    final float q = (float) ((r * r) * (1.0 + epsilon));
21    final float lq = (float) Math.log(q);
22
23    log("p: " + p);
24    log("l: " + l);
25    log("P: " + P);
26    log("L: " + L);
27    log("epsilon: " + epsilon);
28    log("q: " + q);
29    log("lq: " + lq);
30    log("r: " + r);
31    log("lr: " + lr);
32    log("P * q < r * r: " + (P * q < r * r));
33    log("l * q > r * r: " + (l * q > r * r));
34
35    float sum1 = 0.0f;
36    float sum2 = 0.0f;
37    long t = 0;
38    for (t = 0; t < maxT; t++) {
39        float temp = sum1;
40        temp += L;
41        if (temp == sum1) {
42            log("(!) INDISTINGUISHABLE RESULTS AT t: " + t + "; UNSAFE SEQUENCE LENGTH T: " + (t + 2));
43            break;
44        }
45        sum1 += l;
46        sum1 += l;
47        sum2 += l;
48        sum2 += l;
49    }
50    log("SUM1: " + sum1);
51    log("SUM2: " + sum2);
52
53    sum1 += L;
54    sum1 += L;
55    sum1 += L;
56    sum1 += lq;
57    log("(((SUM1 + L) + L) + L) + lq: " + sum1);
58
59    sum2 += L;
60    sum2 += lr;
61    sum2 += L;
62    sum2 += lr;
63    log("(((SUM2 + L) + lr) + L) + lr: " + sum2);
64
65    if (sum1 < sum2)
66        log("P(PATH1) < P(PATH2)");
67    else if (sum1 == sum2)
68        log("P(PATH1) == P(PATH2)");
69    else
70        log("P(PATH1) > P(PATH2)");
71
72    return t;
73 }

```

Wywołanie prezentowanej funkcji pokazuje, że krytyczne T^* wynosi 514. Ścieżka „nieparzysta” wygrywa wówczas błędnie z „parzystą”. Natomiast wywołując to samo dla $T^* - 1$ otrzymujemy poprawne wskazanie przeciwne. Oba te przypadki zaprezentowano poniżej w postaci wypisu z konsoli będącego wynikiem pracy programu z Listingu 6.1.

Suma dla ścieżki „nieparzystej” (SUM2) jest nieprawidłowo większa:

TEST VITERBI (FLOATS) FOR T: INFINITY
p: 1.12535176E-7
l: -16.0
P: 0.9990239
L: -9.765625E-4
epsilon: 4.885196684653037E-4
q: 2.251803E-7
lq: -15.306364
r: 4.744158E-4
lr: -7.653426
P * q < r * r: true
1 * q > r * r: true
(!) INDISTINGUISHABLE RESULTS AT t: 512; UNSAFE SEQUENCE LENGTH T: 514
SUM1: -16384.0
SUM2: -16384.0
((((SUM1 + L) + L) + L) + lq: -16399.307
((((SUM2 + L) + lr) + L) + lr: -16399.31
P(PATH1) > P(PATH2)

Suma dla ścieżki „parzystej” (SUM1) jest prawidłowo większa:

TEST VITERBI (FLOATS) FOR T: 513
p: 1.12535176E-7
l: -16.0
P: 0.9990239
L: -9.765625E-4
epsilon: 4.885196684653037E-4
q: 2.251803E-7
lq: -15.306364
r: 4.744158E-4
lr: -7.653426
P * q < r * r: true
1 * q > r * r: true
SUM1: -16352.0
SUM2: -16352.0
((((SUM1 + L) + L) + L) + lq: -16367.31
((((SUM2 + L) + lr) + L) + lr: -16367.309
P(PATH1) < P(PATH2)

Rozważmy przypadek syntetycznego HMM, dla którego liczby stanów i obserwacji wynoszą odpowiednio $N = 30$ i $M = 1000$. Estymacja parametrów odbywa się na sekwencji długości $T = 1000$ elementów wytworzonych syntetycznie za pomocą wzorów:

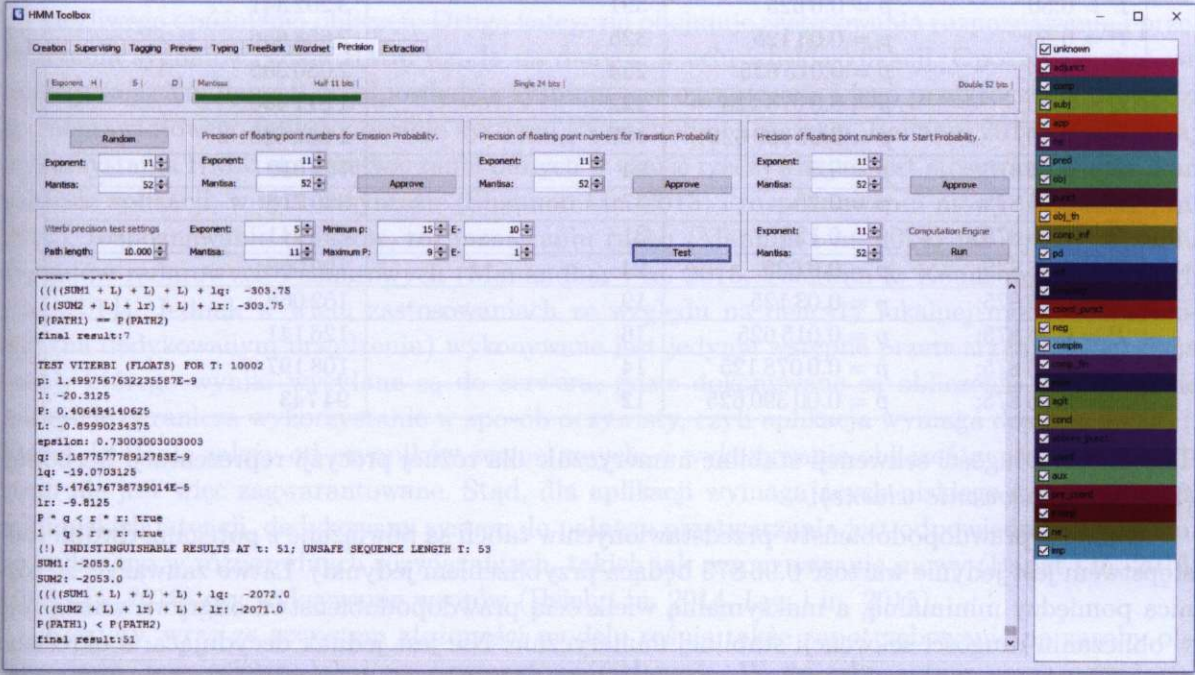
$$\begin{aligned} q_t &= \{t \bmod (t \bmod N) + 1\}, & 1 \leq t \leq T, \\ o_t &= \{t \bmod (t \bmod M) + 1\}, & 1 \leq t \leq T. \end{aligned} \quad (6.15)$$

Prawdopodobieństwa przejścia i emisji uzyskane są przez zliczanie odpowiednio częstości przejść stanów i wystąpień obserwacji na podstawie uzyskanych sekwencji z uwzględnieniem poprawki Laplace’a. Maksymalne P i minimalne p z niezerowych prawdopodobieństw dla obu matryc są następujące:

$$\begin{aligned} P_A &= 0.5396825396825396, & p_A &= 0.006211180124223602, \\ P_B &= 0.0019960079840319, & p_B &= 0.000884173297966401. \end{aligned} \quad (6.16)$$

Znając warunki konieczne dla bezpieczeństwa numerycznego obliczeń w ramach HMM, możliwe jest, określenie maksymalnej długości badanych sekwencji. Dla HMM o zadanych macierzach przejść i emisji, istnieje minimalna precyzja reprezentacji liczb zmiennoprzecinkowych, dla której badanie sekwencji obserwacji o określonej długości T jest numerycznie stabilne. Ponadto, w zakresie dekodowania ścieżek stanów, istnieje maksymalna długość sekwencji obserwacji, na której przeprowadzenie obliczeń (suma i akumulacja logarytmów) nie doprowadzi do przepełnienia i związanej z tym nierozróżnialności wyniku. Poprzez zmniejszenie dokładności reprezentacji liczb zmiennoprzecinkowych z podwójnej (64 bitów) do połówkowej (16 bitów), wszystkie prawdopodobieństwa i odpowiadające im wartości w przestrzeni logarytmicznej zostaną zaokrąglone w celu dopasowania do nowego formatu. Zapewnienie obliczeń stabilnych numerycznie w obniżonej precyzji realizowane jest przez zastosowanie ograniczeń sekwencji z wzoru (6.14). Zatem maksymalna długość badanej sekwencji obserwacji dla przedstawionego przypadku syntetycznego HMM zrealizowanego na połówkowej dokładności reprezentacji liczb zmiennoprzecinkowych jest ograniczona do 147 elementów.

Numerycznie stabilna długość sekwencji obserwacji zależy od zestawu prawdopodobieństw i precyzji reprezentacji liczbowej. Obliczenia w ramach HMM o podwójnej precyzji mogą być traktowane jako numerycznie stabilne. Jednakże, w systemach przyspieszania obliczeń (m.in. realizowanych na GPGPU), gdzie powszechnie stosowane są pojedyncza, a nawet połówkowa precyzja obliczeń, ograniczenia długości dla badanej sekwencji obserwacji muszą być brane pod uwagę. Dlatego też parametry obliczeniowe należy dobrać również w zależności od potrzeb aplikacji, dostępnego sprzętu (silnik obliczeniowy o zadanej precyzji) oraz stosowanego ukrytego modelu Markowa, tak aby bezpieczeństwo obliczeń numerycznych było zagwarantowane. HMM-Toolbox w zakładce „precision“ umożliwia sprawdzenie numerycznej stabilności modelu względem zadanych parametrów obliczeniowych i długości badanej sekwencji obserwacji. Rysunek 6.2 przedstawia przykładowe zastosowanie HMM-Toolbox w celu oceny modelu dla zredukowanej precyzji obliczeniowej.



Rysunek 6.2: Widok zakładki do redukcji reprezentacji liczbowej w ramach aplikacji HMM-Toolbox.

(Źródło: opracowanie własne).

Tabela 6.1 przedstawia zastosowania lematu dla otrzymania numerycznie stabilnych obliczeń względem różnych zestawów prawdopodobieństwa jak i dla różnej precyzji liczb zmiennoprzecinkowych.

Zestaw prawdopodobieństw [p, P]	T^*	
	Półówkowa precyzja (16 bitów)	Pojedyncza precyzja (32 bity)
$P = 0.0078\ 125;$ $p = 0.00\ 390\ 625$	1 927	15 785 253
$P = 0.015\ 625;$ $p = 0.0\ 078\ 125$	2 014	16 491 530
$P = 0.015\ 625;$ $p = 0.00\ 390\ 625$	1 927	15 785 253
$P = 0.03\ 125;$ $p = 0.015\ 625$	1 022	8 371 761
$P = 0.03\ 125;$ $p = 0.0\ 078\ 125$	990	8 102 922
$P = 0.03\ 125;$ $p = 0.00\ 390\ 625$	903	7 396 645
$P = 0.0\ 625;$ $p = 0.03\ 125$	1 690	13 843 467
$P = 0.0\ 625;$ $p = 0.015\ 625$	1 022	8 371 761
$P = 0.0\ 625;$ $p = 0.0\ 078\ 125$	990	8 102 922
$P = 0.0\ 625;$ $p = 0.00\ 390\ 625$	903	7 396 645
$P = 0.125;$ $p = 0.0\ 625$	1 927	15 785 253
$P = 0.125;$ $p = 0.03\ 125$	1 690	13 843 467
$P = 0.125;$ $p = 0.015\ 625$	1 022	8 371 761
$P = 0.125;$ $p = 0.0\ 078\ 125$	478	8 102 922
$P = 0.125;$ $p = 0.00\ 390\ 625$	391	7 396 645
$P = 0.25;$ $p = 0.125$	1 022	8 371 761
$P = 0.25;$ $p = 0.0\ 625$	903	7 396 645
$P = 0.25;$ $p = 0.03\ 125$	666	5 454 859
$P = 0.25;$ $p = 0.015\ 625$	510	4 177 457
$P = 0.25;$ $p = 0.0\ 078\ 125$	478	3 908 618
$P = 0.25;$ $p = 0.00\ 390\ 625$	391	3 202 341
$P = 0.50;$ $p = 0.25$	903	7 396 645
$P = 0.50;$ $p = 0.125$	510	4 177 457
$P = 0.50;$ $p = 0.0\ 625$	391	3 202 341
$P = 0.50;$ $p = 0.03\ 125$	325	2 658 656
$P = 0.50;$ $p = 0.015\ 625$	254	2 080 305
$P = 0.50;$ $p = 0.0\ 078\ 125$	222	1 811 466
$P = 0.50;$ $p = 0.00\ 390\ 625$	187	1 524 620
$P = 0.96\ 875;$ $p = 0.5$	94	762 019
$P = 0.96\ 875;$ $p = 0.25$	47	380 718
$P = 0.96\ 875;$ $p = 0.125$	31	253 241
$P = 0.96\ 875;$ $p = 0.0\ 625$	24	190 068
$P = 0.96\ 875;$ $p = 0.03\ 125$	19	152 068
$P = 0.96\ 875;$ $p = 0.015\ 625$	16	126 141
$P = 0.96\ 875;$ $p = 0.0\ 078\ 125$	14	108 197
$P = 0.96\ 875;$ $p = 0.00\ 390\ 625$	12	94 743

Tabela 6.1: Długości sekwencji stabilne numerycznie dla różnej precyzji reprezentacji liczbowej. (Źródło: *opracowanie własne*).

Wartości prawdopodobieństw przedstawionych w tabeli są powiązane z potęgami dwójki (odstępstwem jest jedynie wartość 0.96 875 będąca przybliżeniem jedynek). Łatwo zauważyć, że różnica pomiędzy minimalną, a maksymalną wielkością prawdopodobieństwa odgrywa ważną rolę w obliczaniu długości sekwencji stabilnej numerycznie. Nie jest jednak decydująca, a określone długości sekwencji pokrywają się pomimo różnego odstępstwa między wartościami prawdopodobieństw. O ile dla pojedynczej precyzji liczb zmiennoprzecinkowych otrzymywane długości nie wydają się być problematyczne, to już dla precyzji półówkowej konieczne jest uprzednie sprawdzenie, czy długość badanej sekwencji obserwacji pozwala na obliczenia stabilne numerycznie. Przekraczanie długości sekwencji (o której mówi lemat) oprócz tego, że może wprowadzać błędy numeryczne, to także w sposób pośredni może psuć dokładność klasyfikacji (w tym czułość, specyficzność, lub inne miary). Eksperyment to potwierdzający znajduje się w podrozdziale 7.8.2.

Rozdział 7

Akceleracja algorytmów uczenia maszynowego

W ramach rozprawy zrealizowane przez autora zostały przyspieszenia sprzętowe algorytmów stosowanych w obliczeniach HMM. Korzyści z przyspieszenia widoczne są zarówno dla systemów czasu rzeczywistego, jak i w dla infrastruktury chmur obliczeniowych. Poniżej przedstawiona została motywacja dla potrzeby realizacji tego typu rozwiązań oraz główne wyzwania w implementacji.

Ze względu na swoją bogatą strukturę matematyczną, ukryte modele Markowa (HMMs) są wykorzystywane w wielu aplikacjach. Większość z tych zastosowań dotyczy zadań rozpoznawania i klasyfikacji wzorców i można je podzielić na dwie główne kategorie w zależności od narzuconych ograniczeń czasowych. Pierwsza kategoria obejmuje ogólną eksplorację danych, gdzie HMM operuje na danych dużych rozmiarów („big data“) w celu znalezienia statystycznie ważnych wzorców, aby wyodrębnić istotne informacje (Mari & Ber 2006, Panuccio i in. 2002). W wielu przypadkach taka analiza może być wykonana w chmurze bez specjalnych ograniczeń co do możliwego opóźnienia obliczeń. Druga kategoria obejmuje zastosowania rozpoznawania i przetwarzania sygnałów, gdzie HMM działa na danych w celu ich klasyfikacji. Opóźnienie czasowe między wejściem danych, a odpowiedzią systemu jest ograniczone i jego przekroczenie prowadzi do nieprawidłowego funkcjonowania systemu (Narasimhan i in. 2006, Behnam 2015). Ten rodzaj wykorzystania HMM do przetwarzania danych w czasie rzeczywistym jest stosowany w szerokim zakresie aplikacji, w tym w syntezie (Shannon i in. 2013) i rozpoznawaniu mowy (Kubanek i in. 2012), rozpoznawaniu obrazów, rozpoznawaniu ruchu (Mannini i in. 2014) jak i przetwarzaniu sygnałów radarowych i sonarowych (Manandhar i in. 2015, Taehwan & Keunsung 2011, Singh i in. 2014). Jednak w wielu zastosowaniach ze względu na deficyty lokalnej mocy obliczeniowej (na dedykowanym urządzeniu) wykonywane jest jedynie wstępne przetwarzanie (ekstrakcja cech), którego wyniki wysyłane są do serwera, gdzie dokonywane są obliczenia HMM. Takie podejście ogranicza wykorzystanie w sposób oczywisty, czyli aplikacja wymaga dostępu do sieci, której działanie zależy od czynników zewnętrznych, a wykonywanie obliczeń w czasie rzeczywistym nie jest więc zagwarantowane. Stąd, dla aplikacji wymagających niskiego czasu obliczeń, jak i niskich latencji, dedykowany system do pełnego przetwarzania jest odpowiednim wyborem, co wykazano w różnorodnych rozwiązaniach, takich jak rozpoznawanie mowy (Bapat i in. 2013, Choi i in. 2010) czy wykrywanie wzorów (Hsieh i in. 2014, Lyu i in. 2015).

Niestety, wraz ze wzrostem złożoności modelu rośnie także zapotrzebowanie na zasoby obliczeniowe, pamięciowe, jak i na przepustowość danych. W związku z tym przy rozważaniu realizacji sprzętowej algorytmów HMM (algorytmy Forward-Backward, Baum-Welch i Viterbi) podejmowana jest próba minimalizowania rozmiaru HMM, przy jednoczesnym zachowaniu stabilności numerycznej obliczeń. Jak wspomniano w pracy (Rabiner 1989) typowe obliczenia związane z HMM szybko wyczerpują dokładność reprezentacji numerycznej. Jest to związane z koniecznością wykonywania długich sekwencji mnożeń wartości (bliskich zera) prawdopodobieństw przejścia lub emisji obserwacji ze stanu. Dlatego też, aby uniknąć tych komplikacji

kluczowe algorytmy są obliczane w dziedzinie logarytmów, z pominięciem stosowania współczynników skalowania (Brown & Golod 2009). Zmniejszenie wielkości modelu HMM można uzyskać poprzez zmniejszenie precyzji reprezentacji liczbowej dla macierzy przejść i emisji, np. do: 32 bitów (pojedyncza precyzja), 16 bitów (połówkowa precyzja), a nawet 8 bitów (ćwiartkowa precyzja). Redukcja ta ma bezpośredni wpływ na model i ogranicza maksymalną długość badanej sekwencji obserwacji. Ilościową wartość tego ograniczenia podaje lemat przedstawiony w rozdziale 6.1. Przy stosowaniu redukcji precyzji zmiennoprzecinkowej, stosowanie się do wniosków płynących z lematu wystarcza na zachowanie bezpieczeństwa numerycznego.

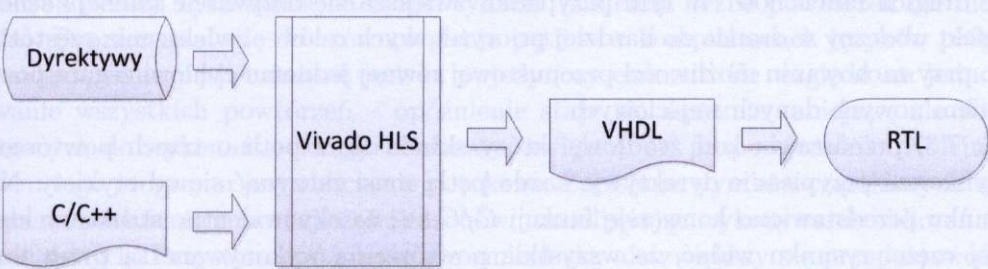
Przyspieszenie i zrównoleglenie algorytmów HMM nie jest zadaniem trywialnym, jednakże dla wielu aplikacji konieczne, zwłaszcza w dziedzinie bioinformatyki (Chrysos & i in. 2012) oraz w systemach wbudowanych (takich jak robotyka, pojazdy autonomiczne czy też internet rzeczy). W ostatnich latach przeprowadzone zostały liczne badania dotyczące przyspieszania algorytmów HMM (Ibrahim i in. 2016, Vargas i in. 2001, Sun i in. 2009, Majumder i in. 2014) z wykorzystaniem FPGA. Większość prac rozwojowo-badawczych skupiają się na dekodowaniu HMM z wykorzystaniem algorytmu Viterbiego (obliczonego zarówno w postaci tradycyjnej lub zlogarytmowanej) oraz na ocenie HMM za pomocą algorytmu Forward-Backward (Churbanov & Winters-Hilt 2008). Mając na celu uzyskanie jak najlepszych wyników wydajności, kwestie dokładności i stabilności numerycznej są często niedoceniane lub całkowicie pomijane. Główne przeszkody w wykorzystaniu HMM w aplikacjach czasu rzeczywistego wymagających niskich latencji są ograniczenia dostępu do zasobów szybkiej pamięci i niestabilność numeryczna obliczeń. Złożona struktura HMM wymaga dziesiątki megabajtów na przechowywanie macierzy przejść i emisji (na przykład dla HMM z 10^2 stanów i 10^5 obserwacji wymaga około 8 MB zasobów pamięciowych). Jest to główną przeszkodą w obliczaniu modelu bezpośrednio w układzie FPGA lub w pamięci podręcznej procesora. Oczywistym rozwiązaniem tego problemu jest zmniejszenie dokładności reprezentacji liczb zmiennoprzecinkowych w obu macierzach. Jednak zmniejszenie precyzji ma istotny wpływ na stabilność numeryczną, zwłaszcza w algorytmach Forward-Backward i Viterbiego, gdzie wiele wartości prawdopodobieństw (głównie wartości bliskie zeru) są mnożone, co może szybko doprowadzić do niedomiaru arytmetycznego (Tatavarty 2011). Możliwym rozwiązaniem tego problemu dla algorytmu Forward-Backward (i dalej w Baum-Welch) jest użycie współczynników skalujących, które pozwalają na przeniesienie obliczeń z dala od niedomiaru. Jednakże współczynniki skalujące muszą być dodatkowo obliczane oraz przechowywane dla całej badanej sekwencji obserwacji, a także do dalszego obliczania algorytmu Baum-Welch. Jest to niekorzystne (przede wszystkim dla implementacji w FPGA), ze względu na dodatkowe wymagania pamięciowe. Z punktu widzenia stabilności numerycznej sensowne jest przeprowadzenie całego obliczenia w dziedzinie logarytmów (jak to ma miejsce dla algorytmu Viterbiego (Rabiner 1989)), gdzie zamiast iloczynu obliczana jest suma logarytmów. Aby efektywnie skorzystać z tego rozwiązania, kosztowne przekształcenia logarytmiczne i wykładnicze można suplementować szybką aproksymacją. W tym celu zrealizowana została specjalna jednostka minimalizująca koszt tych operacji przez jednoczesną aproksymację wielu wartości zarówno logarytmicznych i wykładniczych.

7.1 Synteza wysokiego poziomu

Vivado HLS (ang. *High Level Synthesis*) firmy Xilinx, oferuje środowisko przeznaczone do projektowania układów cyfrowych przy użyciu języka wysoko poziomowego. Próby wykorzystania języków wysokiego poziomu (C/C++) były wielokrotnie podejmowane przez różne instytucje badawcze. Głównym celem tych badań było umożliwienie projektowania FPGA dla specjalistów zaznajomionych z programowaniem sekwencyjnym. Praktyczne wykorzystanie układów FPGA często przysparza wiele trudności dla mało doświadczonych projektantów-programistów, którzy borykają się z licznymi nietypowymi zadaniami: koniecznością pamiętania o prawidłowym formowaniu sygnałów taktowania, uwzględnianiu opóźnień, a nawet ogólnie zrozumienia, że dany operator/znak w języku opisu sprzętu nie zupełnie odpowiada operatorowi/znakowi w języku programowania. Na przykład znaki działań arytmetycznych w popularnych językach opisu

sprzętu HDL (ang. *Hardware Description Language*) wymagają odpowiedniej oprawy. Zależnie od tego, czy operacja dodawania pojawi się w postaci przypisania kombinatorycznego, czy wewnątrz procesu synchronicznego – otrzymamy różne rozwiązania strukturalno-behawioralne. W efekcie, przy projektowaniu na bazie FPGA wkłada się wiele wysiłku w opracowanie automatu, aktywującego funkcje w odpowiednich momentach. Znając odpowiednią kolejność operacji, można zastosować zasady regulujące i zbudować odpowiedni automat. Należy pamiętać, że jego tworzenie można zautomatyzować, poprzez opracowanie odpowiedniego programu analizującego tekst źródłowy napisany w języku wysoko poziomowym. Rozwiązania w postaci kompilatorów generujących z języka C/C++ opis VHDL czy Verilog na poziomie RTL (ang. *Register Transfer Level*), są stosunkowo nowe (Nane i in. 2016). Otrzymane pliki można było zintegrować z projektem na równi z plikami otrzymanymi innymi metodami. Charakterystyki projektów, stworzonych za pomocą narzędzi „C → HDL”, na podstawie kompilacji, zazwyczaj odznaczały się nieco większą ilością źródeł i nieco większą częstotliwością taktowania logiki. Było to efektem tego, że kompilatory rozbiły złożone obliczenia na podstawowe operacje, a każda z tych operacji wprowadzała niewielkie opóźnienie. Połączenie takich elementarnych schematów sterowanych końcowym automatem dawało w rezultacie regularną strukturę rejestrów, przekazujących sobie dane poprzez węzły obliczeniowe, a podobne struktury są właśnie efektywnie realizowane w FPGA. Podobieństwo języków „C → HDL” z językiem programowania C/C++ nie oznacza, że utworzony projekt w FPGA, wykonywać będzie wyliczenia, odpowiadające programowi C/C++ dla procesora. Wybór C/C++ jako podstawy można tłumaczyć jego szerokim rozprzestrzenieniem i ułatwieniem nauki dla specjalistów z dziedziny tworzenia programowania. Jednakże, poza najprostszymi przykładami, kod programów powinien być mocno przetworzony w celu otrzymania rozwiązań sprzętowych, pozwalających na wykorzystanie ich do przyspieszenia interesujących nas obliczeń.

W 2011 roku firma Xilinx ogłosiła wydanie produktu, odnoszącego się do wspomnianej klasy instrumentów, pozwalającego na przekształcenie „C-podobnego” tekstu źródłowego na odpowiadający mu kod w jednym z języków opisu sprzętu. Uzyskany schemat RTL realizuje końcowy automat (tzn. logika, taktowana synchronicznie, co jest korzystne dla współczesnej logiki FPGA), sterujący strumieniami obliczeń. Rezultaty zależą od ograniczeń projektowych (ang. *constraints*), a także, od dyrektyw dla kompilatora. Dyrektywy mogą także ułatwić formowanie interfejsów tworzonego modułu: może być on eksportowany w formie „IP-Core” z interfejsem określonym przez użytkownika, jak również w formie modułu peryferyjnego do instalacji w systemie z procesorem z wykorzystaniem interfejsu AXI (ang. *Advanced eXtensible Interface*). Rysunek 7.1 przedstawia uproszczoną ścieżkę projektowania z wykorzystaniem w Vivado HLS.



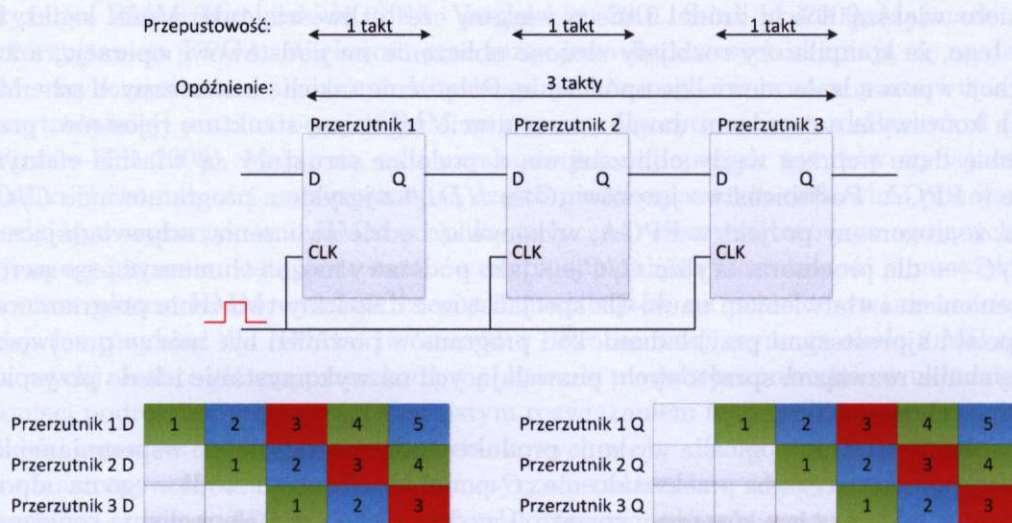
Rysunek 7.1: Uproszczona ścieżka projektowania w Vivado HLS.
(Źródło: opracowane na podstawie (Xilinx 2014)).

7.1.1 Podstawowe informacje o HLS

Dyrektywy HLS znacząco wpływają na wynikowy opis RTL. Z jednego tekstu źródłowego przy pomocy HLS można otrzymać kilka znacznie odróżniających się struktur logicznych. Do analizy tego schematu należy wyjaśnić dwa ważne pojęcia:

- Przepustowość (ang. *throughput*) – mierzona jako opóźnienie w taktowaniu pomiędzy pojawieniem się nowych danych wejściowych.
- Opóźnienie (ang. *latency*) – opóźnienie pomiędzy pojawieniem się danych wejściowych i pojawieniem się odpowiadających im danych wyjściowych.

Pojęcia te zostały zilustrowane na przykładzie prostego schematu logicznego co przedstawia Rysunek 7.2. Na osi czasu widać, że z każdym cyklem zegarowym – dane wejściowe przesuwają się o jeden rejestr, uwalniając przy tym wejście dla przyjęcia nowych danych. W ten sposób przepustowość wynosi jeden takt, ponieważ nowe dane można podawać przy każdym nowym takcie. Opóźnienie jest równe ilości rejestrów w sekwencji. W przedstawionym przypadku – trzy takty.

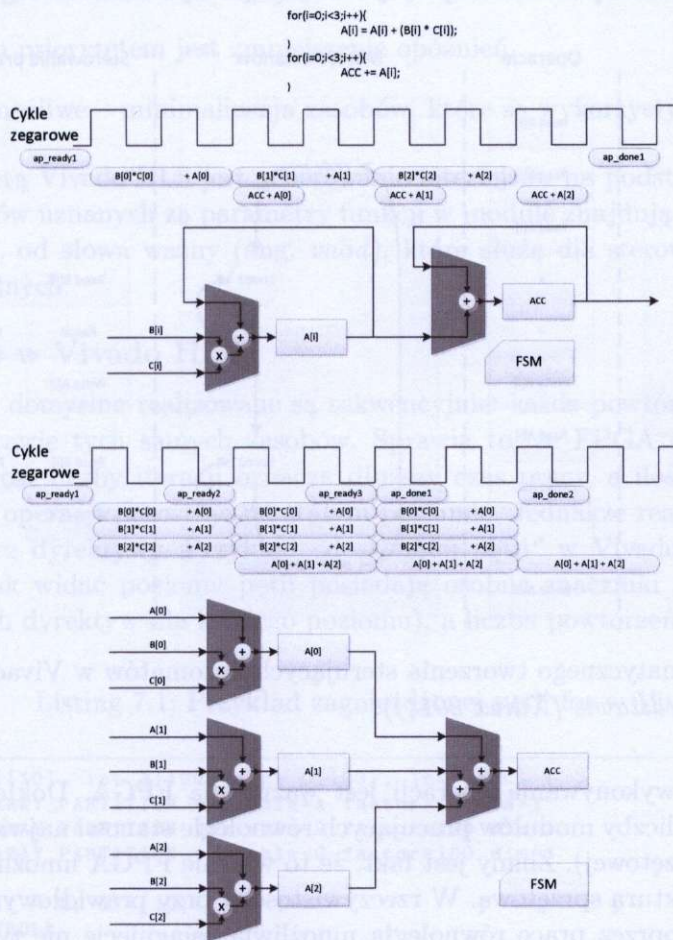


Rysunek 7.2: Pojęcia opóźnienia i przepustowości w FPGA.
(Źródło: opracowanie własne na podstawie (Xilinx 2014)).

Pojęcia te odgrywają ważną rolę przy tworzeniu układów cyfrowych o wysokiej wydajności. Podczas opracowywania złożonego schematu kombinatorycznego częstotliwość taktowania oblicza się za pomocą najdłuższego łańcucha łączącego komponenty synchroniczne. Dlatego oczywistym sposobem na zwiększenie częstotliwości taktowania (czyli skrócenie czasu taktowania) jest rozdzielenie długich łańcuchów. W tym przypadku zwiększa się oczywiście latencja schematu, ale jest to efekt uboczny w drodze do bardziej priorytetowych celów – zwiększenia częstotliwości taktowania, przy zachowaniu możliwości przepustowej równej jednemu cyklowi zegara pomiędzy odczytywaniem nowych danych wejściowych.

Rysunek 7.3. przedstawia kod źródłowy, który składa się z pętli o trzech powtórzeniach. Aby zidentyfikować przypisanie dyrektywy, każda pętla musi zaczynać się od etykiety. Na tym samym rysunku przedstawiono konwersję funkcji C/C++ na ekwiwalentne struktury logiczne. W pierwszej części rysunku widać, że wszystkie powtórzenia wykonywane są przez ten sam sumator/multiplikator. Domyślnie – tworzy się schemat, który odtwarza zachowanie procesora ogólnego przeznaczenia: wszystkie iteracje są wykonywane sekwencyjnie. Takie wykonanie zajmuje najmniejszą powierzchnię w układzie FPGA (tzn. używa najmniejszą ilość zasobów), wymaga jednak większej liczby cykli od rozpoczęcia pracy (sygnalizowane przez `ap_ready1`), aż do otrzymania wyników (tj. ma większą latencję). Ponadto – ponieważ wszystkie iteracje są przeprowadzane za pomocą jednego i tego samego bloku – nie można uruchomić nowego cyklu obliczania do momentu zakończenia bieżącego cyklu (sygnalizowane przez `ap_done1`). Tak więc ten wariant ma małe możliwości przepustowe. W drugiej części rysunku widać, że operacje wykonywane w pętlach zostały zrównoleglone. Takie wykonanie zajmuje największą powierzchnię

w układzie FPGA, ale umożliwia szybsze otrzymanie wyników operacji oraz pozwala na wykonywanie obliczeń w potoku (trzy potoki obliczeniowe sygnalizowane przez ap_ready1, ap_ready2, ap_ready3).

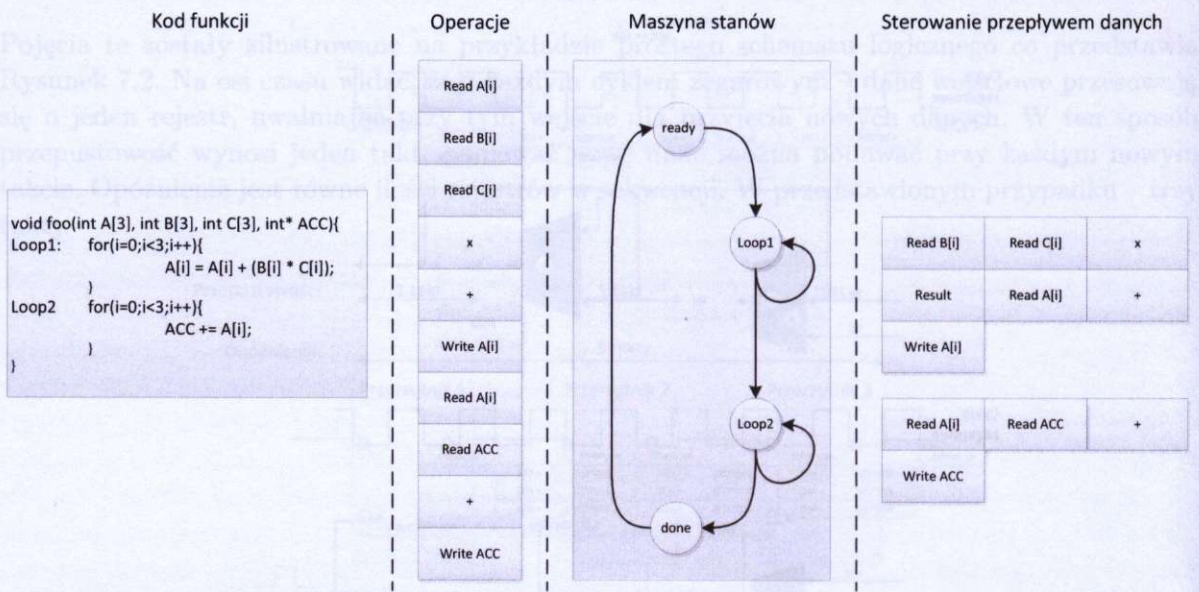


Rysunek 7.3: Podstawy Vivado HLS – sterowanie syntezą za pomocą dyrektyw.
(Źródło: opracowane na podstawie (Xilinx 2014)).

Jeśli jest to konieczne, pętle można rozwinąć (dyrektywa UNROLL), ale wymaga to tego, żeby liczba iteracji była znana na etapie syntezy (zmienna liczba powtórzeń nie może być rozwinięta). Poprzez analogię do zrównoleglenia pętli – w kompilatorach języków programowania każde odgałęzienie pętli uważane jest za pojedynczy element. Ze względu na jednoczesne wykonywanie wszystkich powtórzeń – opóźnienie staje się znacznie mniejsze, ale ilość zasobów wymaganych do wykonania takiego schematu, zwiększa się proporcjonalnie. Kolejnym udoskonaleniem tego programu jest obliczenie potokowe. Zapisanie wyników częściowych w rejestrach, pozwala na zwolnienie znacznej części zasobów, umożliwiając wykonywanie obliczeń na kolejnym zestawie danych. Przy tym opóźnienie nie zmieni się, ale zwiększy się przepustowość. Komórki logiczne w FPGA, a także zasoby blokowe (bloki pamięci i bloki DSP48) posiadają rejestry wejściowe, dlatego bardziej zaawansowana potokowość jest dopuszczalna dla FPGA. Na podstawie przedstawionego przykładu można określić priorytety w rozwoju schematów wysokiej wydajności. Przede wszystkim, należy zwiększyć przepustowość, aż do jednego taktu na wejściu. W tym przypadku wartość nominalną taktu zwiększa się poprzez pogłębienie potokowości, co automatycznie daje efekt uboczny w postaci zwiększenia opóźnienia. Drugorzędny cel to utrzymanie opóźnienia na rozsądnym niskim poziomie, przy czym nie można pozwolić na pojawienie się zbyt wielu stopni potoku, gdyż nie wpływa to pozytywnie na częstotliwości taktowania.

W celu poprawnego wykonania obliczeń należy zapewnić przepływ danych z jednego węzła obliczeniowego do drugiego w odpowiednich momentach. Vivado HLS automatycznie dobiera

względem oryginalnego kodu źródłowego ścieżki danych i sekwencje sterujące (patrz Rysunek 7.4). Przykład ten pokazuje, że automat sterujący obejmuje początkowy stan, stan przetwarzania pierwszej pętli, stan przetwarzania drugiej pętli i stan zakończenia pracy. Jeżeli występują zależności między danymi, to odpowiednie operacje muszą być wykonane szeregowo.



Rysunek 7.4: Zasady automatycznego tworzenia sterujących automatów w Vivado HLS. (Źródło: opracowanie na podstawie (Xilinx 2014)).

Kwestia równoległego wykonywania operacji jest ważna dla FPGA. Dokładniej – możliwość zorganizowania dużej liczby modułów pracujących równolegle stanowi największą przewagę FPGA (jako platformy sprzętowej). Znany jest fakt, że to właśnie FPGA umożliwia wykonanie obliczeń z unikalną architekturą sprzętową. W rzeczywistości – przy prawidłowym użytkowaniu – logika programowalna poprzez pracę równoległą umożliwia osiągnięcie nie tylko najwyższej wydajności, ale także sprawia, że stosunek ceny do wydajności jest lepszy niż w przypadku procesorów sygnałowych i procesorów ogólnego przeznaczenia. Na przykład FPGA z rodziny Kintex-7 już w najprostszej konfiguracji posiada 240 niezależnych bloków, wykonujących operację „mnożenia i akumulacji”. Biorąc pod uwagę, że częstotliwość taktowania dla FPGA Kintex-7 wynosi do 600 MHz, to nawet uwzględniając obniżenie częstotliwości ze względu na nietypowy routing i niepełne wykorzystanie zasobów, spokojnie oczekiwać można wartości 100 GMAC/s. Można sobie wyobrazić, że całkowity koszt procesorów zapewniających podobną wydajność (100 GHz, pod warunkiem, że na każdym takcie będą wykonywane tylko operacje cyfrowego przetwarzania sygnału) będzie znacznie wyższy niż koszt pojedynczego układu FPGA. Na podstawie powyższych informacji można zauważyć, jak bardzo ważnym jest nie tylko sam wybór takiego algorytmu, który może efektywnie wykorzystywać szczególne właściwości FPGA, ale również takie użycie go, aby w projekcie nie pojawiły się sygnały ze zbyt dużym opóźnieniem, ponieważ to właśnie na ich podstawie będzie wyrównywana częstotliwość taktowania. Ważne jest, że tę pracę Vivado HLS wykonuje automatycznie. Ponieważ jednym z parametrów projektu w HLS jest osiągnięcie wymaganej częstotliwości taktowania, środowisko Vivado może ocenić, czy syntetyzowany system wpasowuje się w te parametry. W przypadku pierwszego wariantu, w celu osiągnięcia wysokiej częstotliwości taktowania wymagane jest programowe przetwarzanie potokowe, dlatego cztery linie kodu źródłowego przekształca się w cztery operacje. W związku z tym opóźnienie takiego układu wynosi cztery. Jednak przy przejściu na platformę sprzętową z wyższą wydajnością (np. z „Spartan 6” na „Kintex 7”) opóźnienia podczas wykonywania kolejnych operacji są mniejsze, dlatego na jednym takcie można wykonać dwie operacje. Częstotliwość taktowania jest utrzymywana na wymaganym poziomie, a opóźnienie ograniczono do dwóch.

Tak więc, podczas generowania schematu, Vivado HLS działa w sposób opisany poniżej.

- Priorytetem jest zwiększenie wydajności schematu przy uwzględnieniu ograniczeń projektowych taktowania, zadawanych przez użytkownika.
- Kolejnym priorytetem jest zmniejszenie opóźnień.
- Jeśli to możliwe – minimalizacja zasobów, które są wykorzystywane przez projekt

Ważną zaletą Vivado HLS jest generowanie interfejsów na podstawie sygnałów sterujących. Oprócz sygnałów uznanych za parametry funkcji w module znajdują się również sygnały z przyrostkiem VLD, od słowa ważny (ang. *valid*), które służą dla sterowania wykonania kolejnych bloków funkcyjnych.

7.1.2 Pętle w Vivado HLS

Pętle „for” domyślne realizowane są sekwencyjnie: każde powtórzenie przeprowadza się kolejno na podstawie tych samych zasobów. Sprawia to, że FPGA zachowuje się podobnie do procesora: wzrost liczby iteracji oznacza dłuższy czas pracy, a ilość zasobów potrzebnych do wykonania tej operacji pozostaje na stałym poziomie. Jednakże realizacja pętli może być kontrolowana przez dyrektywy. Przykład „doskonałej pętli” w Vivado HLS pokazany jest przez Listing 7.1. Jak widać poziomy pętli posiadają osobne znaczniki (co pozwala na ustawienie indywidualnych dyrektyw dla każdego poziomu), a liczba powtórzeń jest stała.

Listing 7.1: Przykład zagnieżdżonej pętli for w Vivado HLS.

```
1 void foo(int A[10], int B[100], int C[100], int* ACC) {
2 #pragma HLS ARRAY_PARTITION variable=A factor=10 dim=1
3 #pragma HLS ARRAY_PARTITION variable=B factor=100 dim=1
4 #pragma HLS ARRAY_PARTITION variable=C factor=100 dim=1
5     int i, j;
6     Loop1: for (i = 0; i < 10; i++) {
7 #pragma HLS UNROLL
8         Loop2: for (j = 0; j < 10; j++) {
9 #pragma HLS UNROLL
10             A[i] = A[i] + (B[i * 10 + j] * C[i * 10 + j]);
11         }
12         ACC += A[i];
13     }
14 }
```

Przykład ten doskonale nadaje się do zilustrowania wpływu dyrektyw kompilatora na wyniki syntezy RTL. Jak już wspomniano, dyrektywa UNROLL prowadzi do rozwijania pętli: powtórzenia są wykonywane równolegle na własnym zestawie zasobów sprzętowych. Liczba taktów potrzebnych do kompletnego wykonania pętli zmniejsza się – zwiększając wielkość struktury schematu. W zasadzie ten efekt jest podstawą jednej z głównych zalet FPGA, posiadających dużą ilość zasobów. Tabela 7.1 przedstawia główne wyniki syntezy dla przykładu prezentowanego przez Listing 7.1, przy doborze różnych dyrektyw kompilacji.

Dyrektywy	Opóźnienie taktów	Przerzutniki	Tablice przeglądowe (LUT)	Bloki DSP48E
Bez dodatkowych dyrektyw	922	135	70	4
HLS UNROLL dla wewnętrznego cyklu	142	1 057	310	40
HLS UNROLL dla wewnętrznego i zewnętrznego cyklu	59	2 235	1 856	48
HLS UNROLL dla obu pętli oraz równoległy dostęp do pamięci tablic A, B, C	7	2 887	1 922	400

Tabela 7.1: Wyniki syntezy projektu z listingu 7.1.

(Źródło: *opracowanie własne*).

Wyniki syntezy bez dodatkowych dyrektyw są przewidywalne. Funkcja na pełne wykonanie wymaga łącznie 922 taktów zegarowych. To rozwiązanie ma minimalne zapotrzebowanie sprzętowe. Dzięki ustawieniu dyrektywy HLS UNROLL dla wewnętrznej pętli (z etykietą Loop2), można zaobserwować spodziewany spadek opóźnienia. Działanie to wymaga użycia dodatkowych bloków DSP w liczbie 40. Oczekiwany efekt przyspieszenia można osiągnąć przez zastosowanie dyrektywy HLS UNROLL dla obu poziomów pętli oraz ustalenia równoległego dostępu do pamięci tablic A, B i C. Ilość zasobów znacznie rośnie: do projektu potrzeba już 400 bloków DSP, wyraźnie rośnie także liczba LUT. Jednak opóźnienie wynosiło tylko 7 taktów! Jeśli celem było maksymalne wykorzystanie zasobów FPGA dla skrócenia czasu pracy, to został on osiągnięty. W ten sposób, za pomocą dyrektyw kompilatora można znacząco zmieniać parametry projektu, regulując ilość przydzielonych dla niego zasobów, jak i opóźnienie. Duża elastyczność przy zadawaniu dyrektyw sprawia, że proces ich realizacji w projekcie nie jest prosty i jednoznaczny, ponieważ nie wszystkie kombinacje dyrektyw prowadzą do zadowalających rezultatów. Dlatego też, należy zwracać szczególną uwagę na charakterystyki osiągane przy bieżących ustawieniach i rozważać różne warianty budowy projektu, style programowania i ustawienia dyrektyw kompilacji.

7.1.3 Tablice i praca z pamięcią

Tablice przeglądowe w HLS realizowane są na podstawie zasobów pamięciowych. Jeśli tablica używana jest jako argument funkcji najwyższego poziomu, pamięć rozmieszcza się poza tą funkcją (np. w bloku BRAM). Dostęp do pamięci blokowej możliwy jest w trybie jedno- jak i dwu- portowym. W razie potrzeby pamięć można podzielić na mniejsze bloki, żeby działała na bazie rejestrów i pamięci rozproszonej (ang. *distributed RAM*). Należy zwrócić szczególną uwagę na wykorzystanie pamięci w HLS. Zazwyczaj programista nie traktuje pamięci komputera jako odrębny zasób, ponieważ wszystkie wykorzystywane dane są automatycznie umieszczane w pamięci głównej komputera. Przy optymalizacji programów można mówić o tymczasowym rozmieszczeniu pewnych wartości w rejestrach, uwzględnieniu buforowania, itd. Jednakże dla FPGA możliwość alokacji danych jest znacznie szersza, a pamięć może być fizycznie realizowana przez następujące zasoby sprzętowe:

- rejestry komórek logicznych;
- tablice typu LUT SliceM (pamięć rozproszona);
- bloki pamięci BRAM.

Ponadto dużą zaletą układów FPGA jest bardzo wysoka teoretyczna wydajność podsystemu pamięci jako całości. BRAM jest typową pamięcią statyczną (to znaczy nie jest programowanym blokiem, a fragmentem płytki krzemowej ze sztywnymi połączeniami). Bloki posiadają całkowicie niezależne interfejsy i mogą pracować przy systemowej częstotliwości taktowania. Tak więc, teoretycznie FPGA o dużej objętości mogą zapewnić dużą wydajność, rzędu setek gigabajtów na

sekundę, co pozostawia daleko w tyle interfejsy zewnętrznych pamięci. Jednak najważniejszym problemem w tym przypadku jest to, czy projekt może osiągnąć przewagę nad wieloma równoległe pracującymi blokami BRAM i czy w ogóle wewnątrz układu może być taki potok danych. Ze względu na te czynniki należy dokładnie przemyśleć rozmieszczenie danych w celu zapewnienia takiej ich lokalizacji w zasobie, która pozwoli na jak najlepsze wykorzystanie FPGA. Projektant podczas programowania powinien się upewnić, że wybrane konstrukcje syntaktyczne pozwalają na rozmieszczenia danych w pamięci, którą ma do dyspozycji. Można stwierdzić, że obecnie wyniki kompilatora HLS w dużym stopniu są uzależnione od cech kodu źródłowego i użytych dyrektyw. Programiście znającemu *C/C++*, może się wydawać, że wyniki pracy HLS w wielu przypadkach są dalekie od doskonałości. Klasycznie podczas wytwarzania oprogramowania w językach programowania wysokiego poziomu nie trzeba troszczyć się o fizyczną strukturę pamięci, ponieważ niezależnie od rodzaju tablicy i metody pracy z nimi wydajność pamięci komputera lub mikrokontrolera jest taka sama, przy czym najczęściej ograniczona przez pojedynczy interfejs dostępu. Kwestie dystrybucji danych po obszarach pamięci mogą być istotne dla systemów wielordzeniowych, posiadających skomplikowaną strukturę dostępu do pamięci na fizycznie oddzielnych szynach. W przeciwieństwie do takich systemów dla efektywnej pracy z pamięcią w FPGA należy wybrać jak najlepszy sposób wykorzystania całkowitej wydajności interfejsów niezależnych bloków pamięci. Na przykład, jeśli wszystkie dane, w celu zaoszczędzenia zasobów, zostaną umieszczone w jednym bloku pamięci, wydajność urządzenia będzie ograniczona do dwóch operacji w czasie jednego taktu (ze względu na to, że blok pamięci w FPGA jest dwuportowy). Natomiast jeżeli rozłoży się programowe jednostki w wielu różnych blokach pamięciowych, to w czasie jednego taktu na każdym z tych bloków będzie można wykonać po dwie operacje, a więc możliwe jest zrównoleglenie pracy algorytmu. Podobnie jak i w przypadku bloków DSP, problem koncentruje się wokół zwiększenia ilości operacji w czasie jednego taktu poprzez zaangażowanie w projekt większej liczby zasobów FPGA. Konkretny typ zasobu dla wybranego elementu funkcjonalnego w HLS można ustawić dyrektywą `RESOURCE`. Aby móc elastyczniej zarządzać pamięcią, wykorzystuje się szereg dyrektyw. Dyrektywa `ARRAY_PARTITION` jest używana do podzielenia pojedynczej tablicy na kilka mniejszych. Dyrektywa `RESHAPE` wykonuje odwrotną operację – łączy kilka niezależnych dziedzin danych w jedną tablicę. W połączeniu z poprzednią dyrektywą pozwala sterować rozmieszczeniem danych w fizycznych blokach, zwiększając lub zmniejszając liczbę niezależnych szyn, przy pomocy których uzyskuje się dostęp do danych. Dyrektywa `PACK` grupuje dane, np. elementy struktury, tworząc w ten sposób szerszą szynę pozwalającą na jednoczesny dostęp do wszystkich jej elementów. Stosowanie tych dyrektyw dla różnych obiektów nie jest jednoznaczne. Skrajną sytuacją jest słaba wydajność szyn dla przesyłu danych i przeciążenie FPGA niezależnymi blokami, powodujące nadmierne zużywanie zasobów. Wybór optymalnej organizacji rozmieszczenia danych jest różny dla każdego projektu. Elastyczność architektury FPGA i duża liczba wariantów (z których nie wszystkie są porównywalne pod względem wydajności) nie pozostawia złudzeń, że kompilator samodzielnie wybierze optymalny sposób rozmieszczenia danych dla dowolnych operacji, które programista chce wykonać.

7.1.4 Użycie języków wysokiego poziomu

W Vivado HLS można skorzystać z wielu języków wysokiego poziomu, głównie *C/C++*, które choć mają podobną składnię, to z formalnego punktu widzenia są różnymi językami, na co zwraca uwagę wielu autorów literatury z dziedziny programowania. Ważną różnicą jest wspieranie tzw. typów o „arbitralnej precyzji”, czyli typów danych o dedykowanej wielkości bitowej. Różnica ta jest uważana za tak ważną, ponieważ podczas klasycznego programowania mamy do czynienia z procesorem przetwarzającym dane o stałej rozdzielczości bitowej określonej przez architekturę tego procesora. Zmiana bitowości z punktu widzenia programu jest możliwa, jednak rzadko ma sens. W przeciwieństwie do powyższego podczas projektowania układów cyfrowych projektant ma możliwość jednoznacznie określić precyzję bitową przetwarzanych danych, która najlepiej pasuje do realizowanego zadania. Typy, które mają wyraźnie określoną liczbę bitów, pozwalają wykorzystać tylko te zasoby sprzętowe, które są wymagane do wykonania zadania.

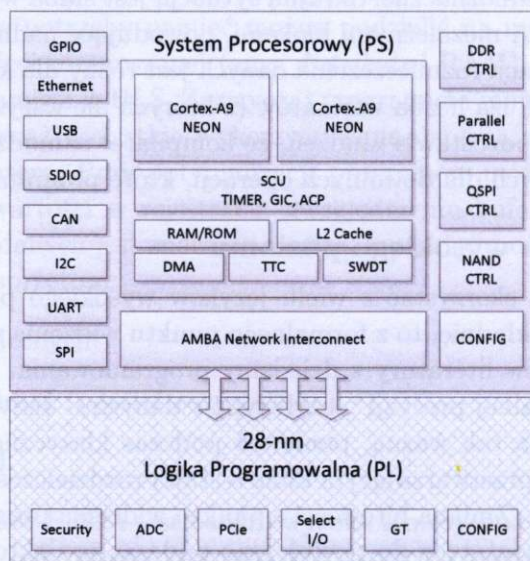
Na przykład biorąc pod uwagę architekturę bloku DSP48E widać, że taki blok jest w stanie pomnożyć 18-bitową liczbę z 25-bitową. Jeżeli więc zadeklarowana szerokość bitowa mnożonych współczynników nie będzie na odpowiednim dla zadania poziomie, tylko zadana domyślnie 32 (automatycznie wskutek wykorzystania typu `int`), to dla utworzonej struktury logicznej potrzebne będą dodatkowe zasoby sprzętowe. Vivado HLS pozwala syntetyzować wiele konstrukcji `C/C++`, pod warunkiem, że ich parametry są znane w czasie kompilacji. Dlatego też to projektant/programista odpowiedzialny jest za parametry architektury obliczeniowej. Warto podkreślić, że jeżeli kryterium efektywności jest czas pracy projektanta, a nie efektywne wykorzystanie zasobów, to użycie typów danych domyślnie wspieranych przez HLS przyniesie pożądany wynik przy krótszym czasie implementacji. Synteza opisów behawioralnych, które ujawniają się dopiero w trakcie pracy układu (ang. *run-time*) z oczywistych powodów nie jest wspierana, jak również nie są syntetyzowane:

- funkcje dynamicznej alokacji pamięci (`malloc()` / `free()`);
- operacje wejścia/wyjścia (`printf()` / `scanf()`);
- wywołania systemowe (ang. *system call*).

ia ta nadaje eleganckie i skuteczne ramy dla realizacji algorytmów HMM. Standardowo w algorytmach programowania dynamicznego dla HMM w każdym kroku czasowym wykonywana jest pętla po wszystkich stanach i dla każdego z nich dokonywane są pewne obliczenia (zwykle sumowanie lub poszukiwanie maksimum). Stąd, oczywistym sposobem na przyspieszenie obliczeń jest przeprowadzenie obliczenia dla wszystkich stanów jednocześnie. Wtedy stopień zrównoleglenia zależy od liczby stanów w modelu Markowa. Najczęściej jednak pętla po wszystkich stanach jest zagnieżdżona w innej pętli po stanach lub iteracji po sekwencji obserwacji. W przypadku gdy zależności wynikające z zagnieżdżenia nie pozwalają na zrównoleglenie algorytmu, to podejmowana jest próba przyspieszenia pętli (na przykład przez przetwarzanie potokowe).

7.1.5 System Procesora (PS) i Logika Programowalna (PL)

Przedstawione rozwiązanie dedykowane jest dla platform typu „System-On-Chip” będących połączeniem typowego procesora z układem FPGA (patrz Rysunek 7.5), gdzie aplikacja HMM wykorzystuje akcelerację sprzętową zrealizowaną w logice programowalnej, a typowe funkcje ogólnego przeznaczenia są nadal wykonywane na CPU.



Rysunek 7.5: Zynq – programowalny układ scalony będący połączeniem procesora i FPGA. (Źródło: opracowanie na podstawie (Xilinx 2017b)).

System procesora zapewnia ogólną infrastrukturę interfejsów komunikacji (Ethernet, HMI, itp.) i jest odpowiedzialny za wstępne przetwarzanie i dekodowanie słownika kodów obserwacji, zaś główne algorytmy HMM, takie jak Viterbi, Forward-Backward i Baum-Welch, obliczane są sprzętowo w dedykowanej strukturze logiki programowalnej. Ze względu na mnogość i ewentualne rozmiary modeli są one przechowywane w pamięci DDR, gdzie w kontenerze modelu HMM zawarte są podstawowe informacje takie jak prawdopodobieństwa macierzy przejść i emisji oraz słownik dekodowania obserwacji wraz z odpowiednim mapowaniem. Ponadto w wielu zastosowaniach opartych o modele HMM słownik dekodowania obserwacji ma od kilku tysięcy do kilkudziesięciu tysięcy elementów (Anisha & C.Sunitha 2015). W typowej aplikacji modelu HMM wyróżnić można następujące etapy przetwarzania:

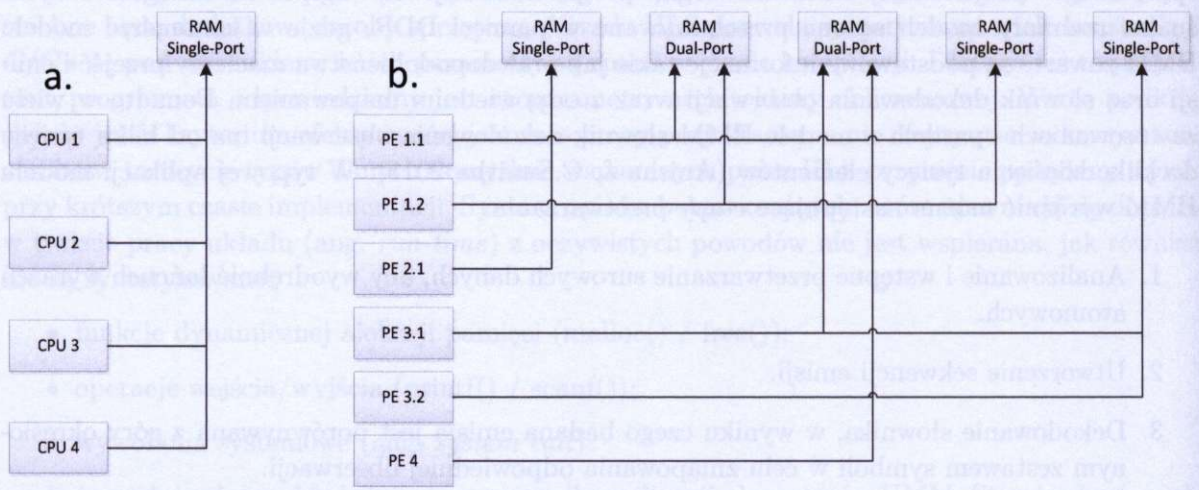
1. Analizowanie i wstępne przetwarzanie surowych danych, aby wyodrębnić łańcuch wyrażen atomowych.
2. Utworzenie sekwencji emisji.
3. Dekodowanie słownika, w wyniku czego badana emisja jest porównywana z góry określonym zestawem symboli w celu zmapowania odpowiedniej obserwacji.
4. Jeżeli celem aplikacji jest zbadanie sekwencji obserwacji, wybierany jest algorytm Viterbigo. Jeżeli celem aplikacji jest nauczanie HMM, wybierany jest algorytm Bauma-Welcha.
5. Ładowanie modelu HMM do „silnika obliczeniowego“.
6. Ładowanie sekwencji obserwacji do „silnika obliczeniowego“.
7. Wykonywanie algorytmu w „silniku obliczeniowym“.
8. Pobranie wyników obliczeń.

Wszystkie etapy z wyjątkiem etapu siódmego są specyficzne dla aplikacji i powinny być wykonywane przez system procesora. Etap 7 można przeprowadzić poza systemem procesora. „Silnik obliczeniowy“ nie definiuje sposobu przetwarzania, tak więc obliczenia mogą być wykonane zarówno przez CPU, system zewnętrzny (GPGPU), jak i w logice programowalnej. Dla aplikacji wymagających niewielkich latencji i przetwarzania w czasie rzeczywistym najbardziej odpowiednim sposobem obliczeń jest logika programowalna. Efektywna sprzętowa realizacja algorytmów HMM wymaga bezpośredniego dostępu do obu macierzy (prawdopodobieństw przejścia i emisji) czyli, parametry modelu HMM należy wprowadzić do FPGA (etap 5). Dalsze obliczenia są wykonywane równolegle dla każdego stanu przez dedykowaną jednostkę przetwarzania stanów SPU (ang. *State Processing Unit*), która operuje na przynależnych do niej wektorach prawdopodobieństw przejść i emisji. Rysunek 7.9. przedstawia ogólną ideę architektury obliczeniowej, gdzie obliczenia związane z modelem HMM, są wykonywane przez szereg jednostek SPU wspomaganych przez jednostkę aproksymacji funkcji logarytmicznych i wykładniczych.

7.2 Dostęp do pamięci, pamięć wieloportowa

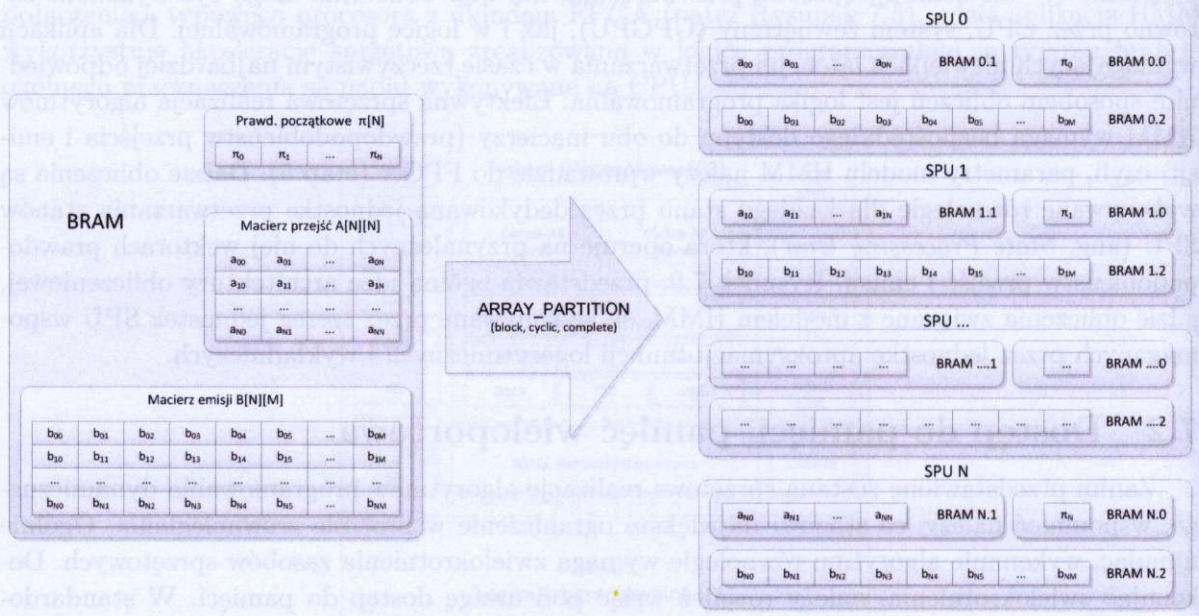
Zanim przedstawione zostaną sprzętowe realizacje algorytmów programowania dynamicznego, wspomnieć należy, co stanowi największe ograniczenie w procesie zrównoleglania. Ogólnie ujmując, wykonanie algorytmu równolegle wymaga zwielokrotnienia zasobów sprzętowych. Dokonując zwielokrotnienia, należy również wziąć pod uwagę dostęp do pamięci. W standardowych rozwiązaniach dostęp do pamięci realizowany jest jednowątkowo z wykorzystaniem szyny adresowej. Adresowana komórka może być zapisana lub odczytana. Dokonując zrównoleglenia samego rdzenia obliczeniowego, a pozostawiając jednowątkowy dostęp do pamięci, prędzej czy później dojdzie do blokowania zasobów pamięciowych i kolejkowania żądań dostępu. Taka sytuacja występuje często w systemach procesorowych ogólnego przeznaczenia (patrz Rysunek 7.6 a).

Wykorzystując jednak logikę programowalną FPGA, możliwe jest utworzenie dedykowanej heterogenicznej struktury obliczeniowej wyposażonej w wiele różnych wyspecjalizowanych pamięci wieloportowych (patrz Rysunek 7.6 b).



Rysunek 7.6: Dostęp do pamięci jedno- i wieloportowej.
(Źródło: opracowanie własne).

Modelowanie struktur pamięciowych również nie należy do trywialnych. Dlatego nie do przecenienia są mechanizmy wbudowane w syntezę wysokiego poziomu pozwalające na zarządzanie dostępem do macierzy wielowymiarowych za pomocą dyrektyw `ARRAY_PARTITION` i `ARRAY_RESHPING`. Stosując wskazane metody optymalizacji dostępu do pamięci, opisane w (Xilinx 2017a), zaproponowano rearanżację macierzy (przechowujących parametry modelu HMM) z przeznaczeniem pod wydajną sprzętową realizację w FPGA. Rysunek 7.7 przedstawia modyfikację dostępu do macierzy wielowymiarowej mapowane na pamięci wieloportowe zlokalizowane w jednostkach przetwarzania stanów dla modelu HMM.



Rysunek 7.7: Modyfikacje dostępu do macierzy wielowymiarowej mapowanej na pamięci wieloportowe.
(Źródło: opracowanie własne).

7.3 Sprzętowa realizacja algorytmu Bauma-Welcha D&C

Algorytm Bauma-Welcha oparty jest na trzech powiązanych ze sobą krokach proceduralnych:

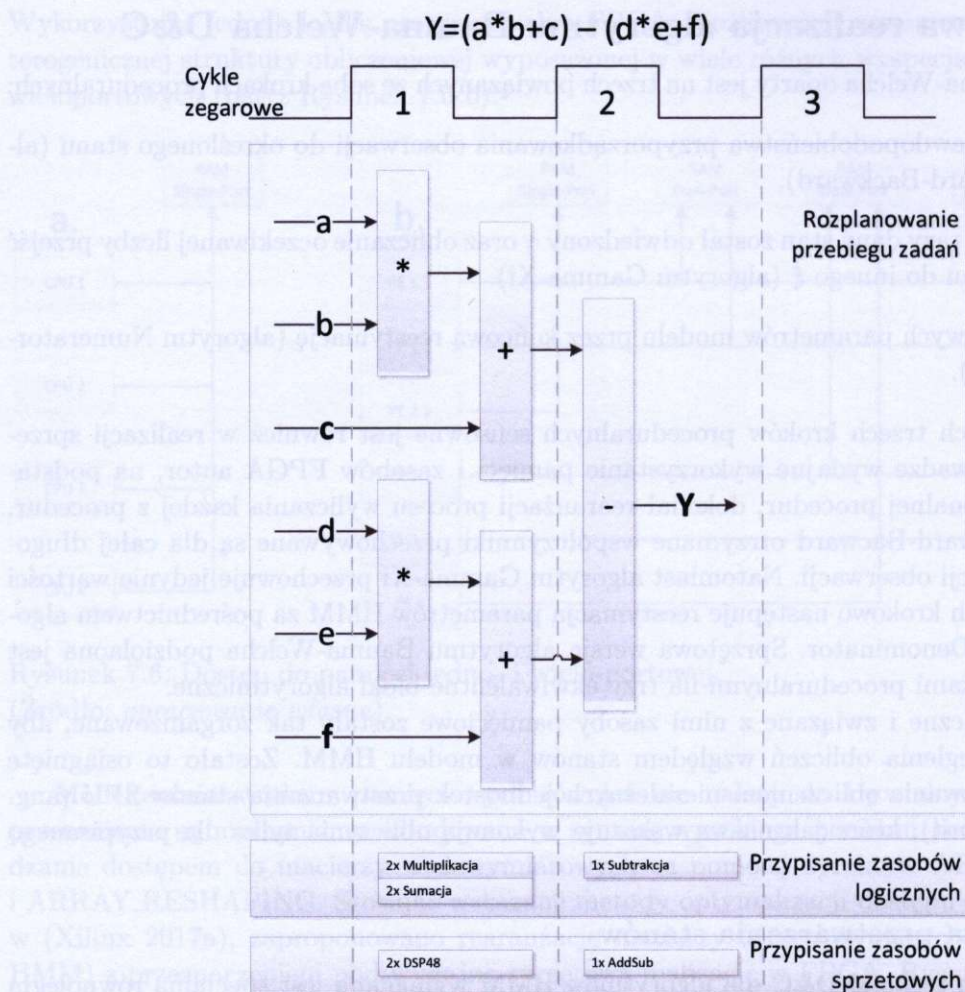
1. Obliczanie prawdopodobieństwa przyporządkowania obserwacji do określonego stanu (algorytm Forward-Backward),
2. Obliczanie ile razy dany stan został odwiedzony γ oraz obliczanie oczekiwanej liczby przejść z danego stanu do innego ξ (algorytm Gamma-Xi)
3. Obliczenie nowych parametrów modelu przez końcową reestymację (algorytm Numerator-Denominator).

Wyróżnienie tych trzech kroków proceduralnych sensowne jest również w realizacji sprzętowej. Mając na uwadze wydajne wykorzystanie pamięci i zasobów FPGA autor, na podstawie analizy funkcjonalnej procedur, dokonał rearanżacji procesu wyliczania każdej z procedur. W algorytmie Forward-Bacward otrzymane współczynniki przechowywane są dla całej długości badanej sekwencji obserwacji. Natomiast algorytm Gamma-Xi przechowuje jedynie wartości cząstkowe, z których krokowo następuje reestymacja parametrów HMM za pośrednictwem algorytmu Numerator-Denominator. Sprzętowa wersja algorytmu Bauma-Welcha podzielona jest więc zgodnie z krokami proceduralnymi na trzy ekwiwalentne bloki algorytmiczne.

Bloki algorytmiczne i związane z nimi zasoby pamięciowe zostały tak zorganizowane, aby umożliwić zrównoleglenia obliczeń względem stanów w modelu HMM. Zostało to osiągnięte przez przyporządkowanie obliczeniom niezależnych jednostek przetwarzania stanów SPU (ang. *State Processing Unit*), które jak nazwa wskazuje wykonują obliczenia tylko dla przypisanego im stanu.

7.3.1 Jednostka przetwarzania stanów

Przy stosowaniu techniki D&C dla algorytmów HMM wymagana jest specjalna równoległa architektura obliczeniowa. Głównym elementem przetwarzania takiej architektury jest jednostka przetwarzania stanów SPU odpowiedzialna za obliczenia na poziomie stanu, gdzie wykonywane są liczne operacje zmiennoprzecinkowe, a przede wszystkim szybkie przekształcenie logarytmiczne i wykładnicze. Wartości prawdopodobieństw w macierzach przejścia i emisji są przechowywane lokalnie w SPU w postaci zlogarytmowanej. Liczba SPU określa stopień zrównoleglenia obliczeń, gdzie górną granicę określa złożoność modelu Markowa (czyli liczba stanów). W obliczeniach wykorzystywana jest multipleksacja czasowa zasobów, co oznacza, że pełne obliczenie algorytmu zostanie wykonane na dostępnej liczbie jednostek przetwarzania stanów. Jeżeli jednostek przetwarzania stanów jest co najmniej tyle co stanów w przetwarzanym modelu, to osiągnięty jest najwyższy stopień zrównoleglenia dla tego modelu. Jeżeli natomiast jednostek przetwarzania stanów jest k -krotnie mniej niż stanów w przetwarzanym modelu to zasoby zostaną k -krotnie powtórnie (sekwencyjnie) wykorzystane, aż algorytm zostanie w pełni przeliczony. Z punktu widzenia efektywności obliczeń podział liczby stanów w przetwarzanym modelu przez liczbę jednostek przetwarzania nie powinien zwracać reszty.



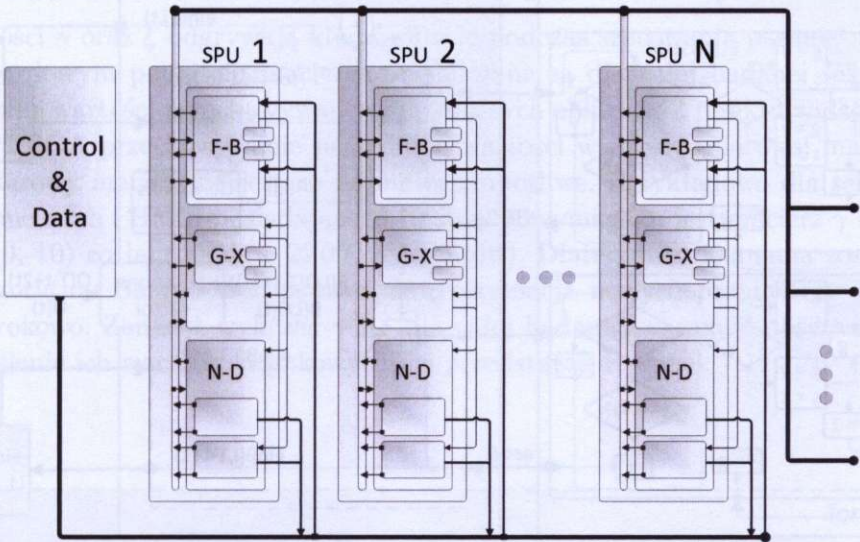
Rysunek 7.8: Przypisanie wykonywanych zadań do zasobów sprzętowych.
(Źródło: opracowane na podstawie (Xilinx 2014)).

SPU realizuje trzy bloki obliczeń algorytmicznych: Forward-Backward, Gamma-Xi i końcowy Numerator-Denominator, przy czym wykorzystuje osiem jednostek zmiennoprzecinkowych oraz dwa niezależne interfejsy dostępu do jednostki MLEAU. Za efektywne wykorzystanie sprzętu odpowiedzialne są mechanizmy alokacji zasobów wbudowane w Vivado. Na poziomie kompilacji LLVM (ang. *Low Level Virtual Machine*) zastosowane są różne metody optymalizacji, takie jak propagacja stałych, eliminacja wspólnych pod wyrażeń i globalnej numeracji zmiennych (Zuo i in. 2013). Na podstawie wyniku wstępnej kompilacji wbudowany w Vivado planista ustala użycie zasobów logicznych dla każdego zadania z uwzględnieniem dyrektyw projektanta oraz zadanego czasu trwania zadania. W pierwszej kolejności system próbuje zastosować zadane dyrektywy, jeżeli jednak w wyniku interpretacji behawioralnego opisu HLS odkryte zostaną zależności uniemożliwiające zastosowanie dyrektyw (np. UNROLL dla pętli) kompilator ma prawo anulować dyrektywę. Kolejnym krokiem jest przypisanie wykonywanych zadań do zasobów sprzętowych, co przedstawia Rysunek 7.8.

W każdym cyklu obliczeniowym sprawdzane jest, które zasoby logiczne są wykorzystane w celu realizacji zadania. Zasoby logiczne mapowane są na sprzęt wyłącznie wtedy, jeżeli w danym cyklu obliczeniowym występuje realne wykorzystanie sprzętu. W miarę możliwości zasoby sprzętowe są współdzielone i wykorzystywane ponownie.

Dlatego też w jednostce przetwarzania stanów w danej chwili wykonywany jest tylko jeden wybrany blok algorytmiczny. Takie podejście pozwala na sensowne współdzielenie zasobów sprzętowych. Ponadto jednostki SPU wykonują obliczenia synchronicznie dla całego modelu. Blok algorytmiczny Forward-Backward jest aktywny w celu wyliczenia wartości prawdopodobień-

stwa prefiksowego α i sufiksowego β dla całej badanej sekwencji obserwacji. Następnie w bloku Gamma-Xi wyliczane są cząstkowe wartości γ i ξ , które to dostarczane są do bloku Numerator-Denominator. Rysunek 7.9 przedstawia ogólną koncepcję przetwarzania HMM z wykorzystaniem SPU.

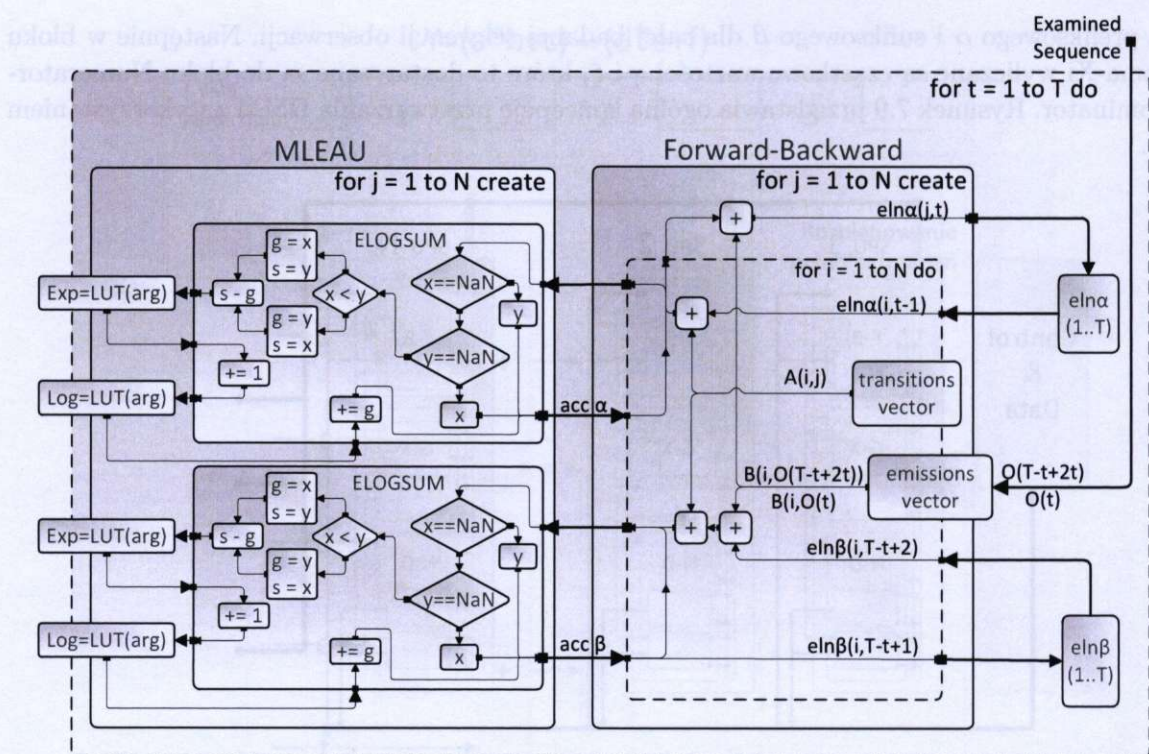


Rysunek 7.9: Ogólna koncepcja przetwarzania HMM z wykorzystaniem SPU.
(Źródło: opracowanie własne).

7.3.2 Blok Forward-Backward D&C

Realizacja algorytmu Forward-Backward w dziedzinie logarytmów nie wymaga współczynników skalowania (znanych z tradycyjnej wersji tego algorytmu), ponieważ w dziedzinie logarytmów obliczenia są natywnie stabilne numerycznie i nie ma potrzeby stosowania dodatkowych ulepszeń. Warto zauważyć, że bez współczynników skalowania obliczanie do przodu i od tyłu mogą być wykonywane równocześnie. Z punktu widzenia sprzętowej implementacji algorytmu współczynniki skalowania wymagają nie tylko dodatkowego nakładu obliczeń, ale także muszą być przechowywane w pamięci FPGA dla całej badanej sekwencji obserwacji do dalszych obliczeń algorytmu Bauma-Welcha.

Rysunek 7.10) przedstawiona diagram wizualizacji bloku algorytmicznego Forward-Backward D&C. Zarówno t jak i kolejne wizualizacje bloków algorytmicznych w spójny sposób przedstawiają najważniejsze cechy proponowanej struktury obliczeniowej. Na rysunkach bloków algorytmicznych pokazane są nie wszystkie sygnały sterujące. Ponadto, linie przerywane przedstawiają iteracyjne wykorzystanie zasobów (np. $\text{elog } \alpha(i, t - 1)$ jest dodawane N razy, względem i inkrementowanego również do N). Natomiast bloki posiadające w nagłówku zapis „for j=1 to N create” przedstawiają zasoby zwielokrotnione N razy. Wszystkie nazwy sygnałów są zgodne z tymi, które zastosowano w pracy (Mann 2006) (np. $\text{elog } \alpha(1..T)$ odnosi się do wektora przechowującego rozszerzone wartości logarytmiczne α). Warunki decyzyjne (związane z ustaleniem odpowiedniej kolejności argumentów oraz wykryciem „Not a Number”) przedstawione dla każdego bloku *ELOGSUM* mają zasadnicze znaczenie dla stabilności numerycznej rozszerzonej arytmetyki logarytmicznej. Wartości widoczne w bloku *ELOGSUM* oznaczają odpowiednio x – pierwszy składnik sumy, y – drugi składnik sumy, g – zmienna przechowująca większy składnik, s – zmienna przechowująca mniejszy składnik oraz acc – oznacza akumulację wyniku. Ponadto widoczne wartości macierzy A , B i O oznaczają odpowiednio parametry modelu HMM oraz badaną sekwencję obserwacji. Symbolicznie (jako $\text{Log}=\text{LUT}(\text{arg})$ oraz $\text{Exp}=\text{LUT}(\text{arg})$) przedstawione zostały również operacje aproksymacji wykonywane w MLEAU.



Rysunek 7.10: Wizualizacja bloku algorytmicznego przeznaczonego pod obliczenia związane z algorytmem Forward-Backward D&C.
(Źródło: opracowanie własne).

Listing 7.2 przedstawia opis HLS dla etapu indukcji w algorytmie Forward-Backward. Operacje w pętli oznaczonej etykietą LOOP_SPU wykonywane są równolegle w jednostkach przetwarzania stanów SPU, których najważniejszym zadaniem jest przyspieszenie sumy logarytmów, co obejmuje operacje arytmetyczne w dziedzinie logarytmów i przejście z argumentu przez przekształcenie wykładnicze, a następnie przekształcenie logarytmiczne. Ta kosztowna obliczeniowo operacja jest wykonywana przy użyciu MLEAU wykorzystywanej w bloku ELOGSUM (ang. *Extended Logarithm Sum*). Ponadto w bloku Forward-Backward obliczany jest rozszerzony logarytm iloczynu prawdopodobieństw przejścia i emisji (wykorzystując wcześniej obliczone odpowiedniki w dziedzinie logarytmów). Wynik przechowywany jest w tablicach elnf i elnb.

Listing 7.2: Opis HLS dla etapu indukcji w algorytmie Forward-Backward D&C.

```

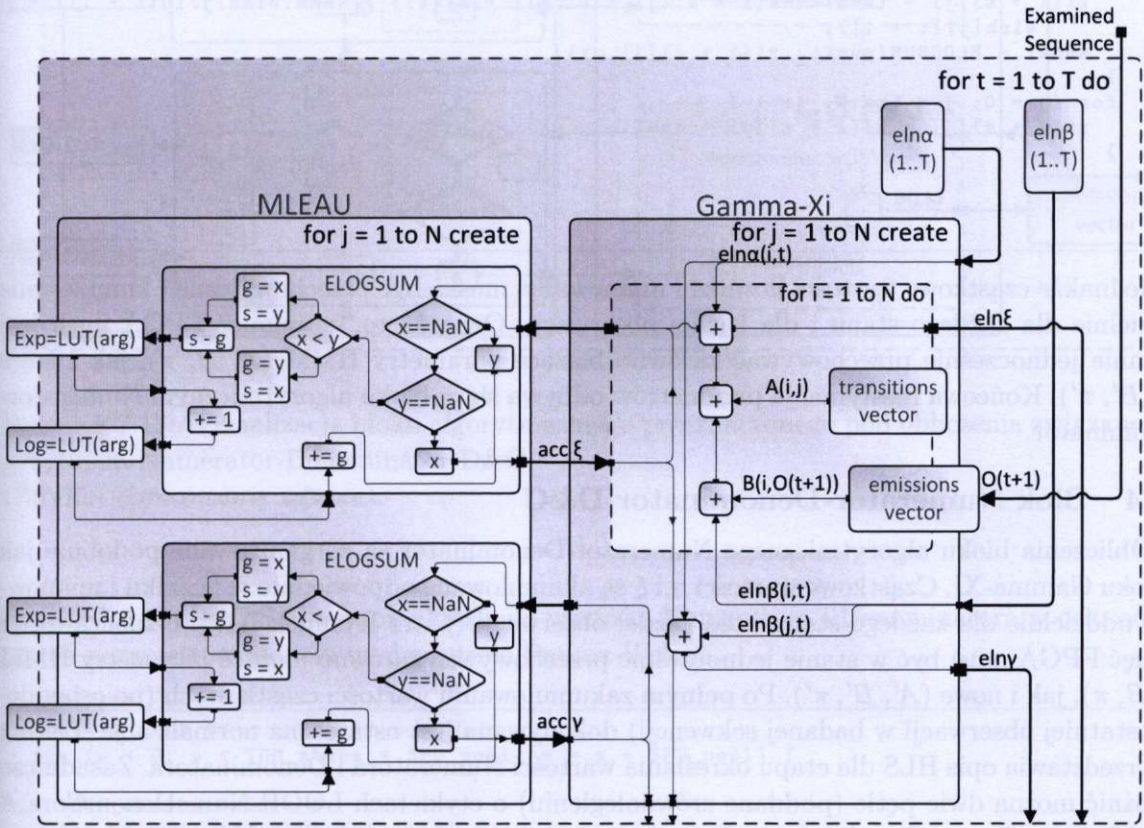
1  for (t = 0; t < T - 1; t++) {
2      for (s = 0; s < TDM_steps; s = s + SPU_pool) {
3          LOOP_SPU:
4              for (i = 0; i < SPU_pool; i++) {
5              #pragma HLS UNROLL
6                  logalpha[i + s] = NaN;
7                  logbeta[i + s] = NaN;
8                  for (j = 0; j < hmm.N; j++) {
9
10                     sumalpha[i + s] = elnf[j][t - 1] + hmm.elnA[i + s][j];
11                     sumbeta[i + s] = hmm.elnA[i + s][j] + hmm.elnB[j][O[T - t + 2]] + elnb[j][T - t + 2];
12                     logalpha[i + s] = ELOGSUM(logalpha[i + s], sumalpha[j], i);
13                     logbeta[i + s] = ELOGSUM(logbeta[i + s], sumbeta[j], i);
14                 }
15                 elnf[i + s][t] = logalpha[i + s] + hmm.elnB[i + s][O[t]];
16                 elnb[i + s][T - t + 1] = logbeta[i + s];
17             }
18         }
19     }

```


Blok algorytmiczny produkuje wartości α i β dla każdego stanu względem badanej sekwencji obserwacji. Wektory α i β są przechowywane w FPGA do dalszych obliczeń przeprowadzanych przez blok algorytmiczny Gamma-Xi.

7.3.3 Blok Gamma-Xi D&C

Wartości γ oraz ξ odgrywają kluczową rolę podczas szacowania parametrów modelu HMM. W standardowym podejściu macierze te obliczane są dla całej badanej sekwencji obserwacji, a otrzymane wartości przechowywane są do dalszych obliczeń. Z punktu widzenia wykorzystania zasobów FPGA przechowywanie pośrednich wartości w dwuwymiarowej macierzy γ , jak i dla trójwymiarowej macierzy ξ jest co najmniej kłopotliwe. Przykładowo dla sekwencji obserwacji o 200 elementach i HMM posiadającego 10 stanów wymagana jest macierz γ (200, 10) i macierz ξ (200, 10, 10) co łącznie daje 22 000 elementów). Dlatego też dokonana została reorganizacja pracy algorytmu Bauma-Welcha taka, że reestymacja nowych parametrów HMM następować będzie krokowo. Zamiast wyliczać γ i ξ dla całej badanej sekwencji obserwacji, wystarczające jest określenie ich wartości cząstkowych, co przedstawia Rysunek 7.11.



Rysunek 7.11: Wizualizacja bloku algorytmicznego przeznaczonego pod obliczenia związane z algorytmem Gamma-Xi D&C.
(Źródło: opracowanie własne).

Listing 7.3 przedstawia opis HLS dla etapu określania wartości Gamma i Xi. W każdej kolejnej chwili czasowej t obliczane i normalizowane są jedynie cząstkowe wartości γ i ξ względem aktualnie badanej obserwacji o_t i na ich podstawie, dokonywane jest również cząstkowe wyliczenie licznika i mianownika w bloku Numerator-Denominator.

Listing 7.3: Opis HLS dla etapu określania wartości Gamma i Xi w algorytmie Bauma-Welcha D&C.

```

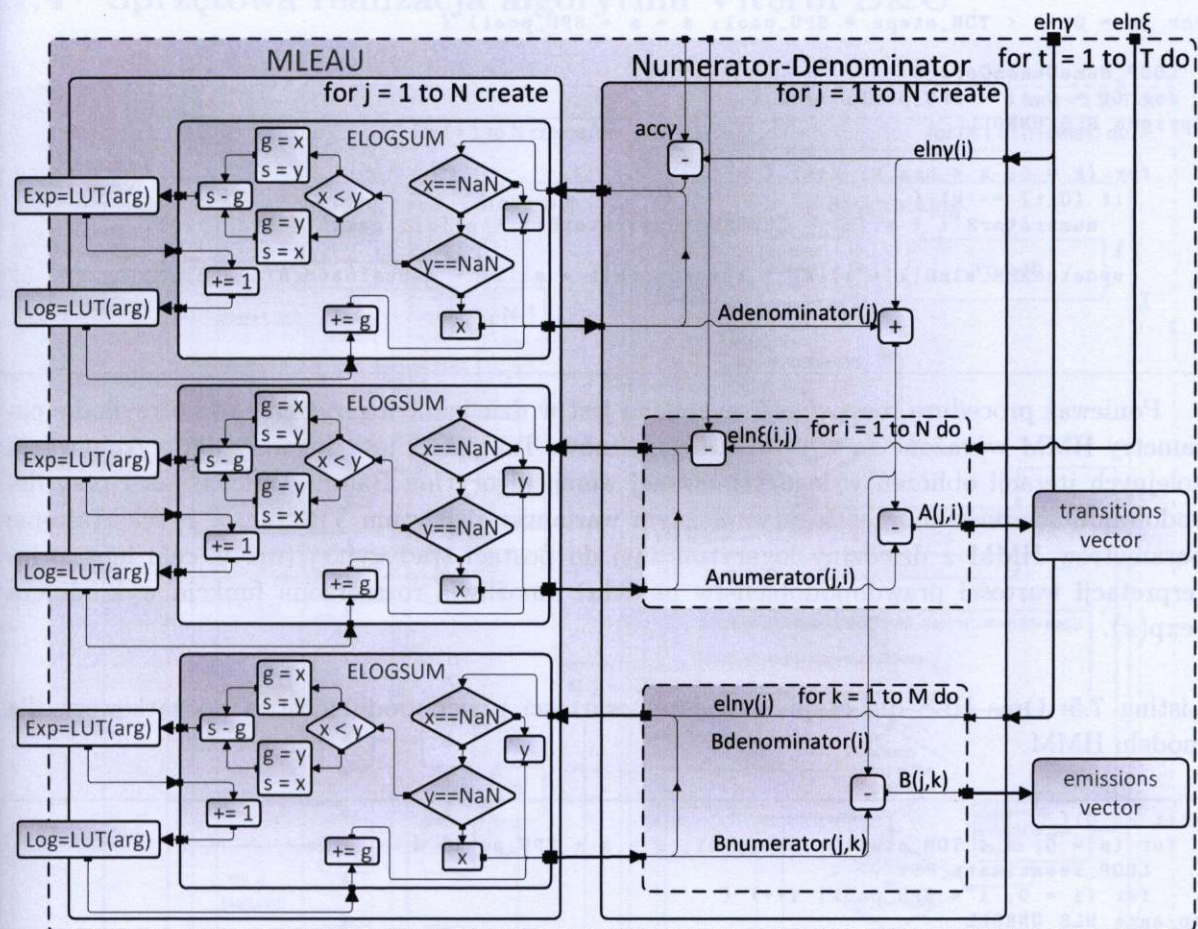
1  for (s = 0; s < TDM_steps * SPU_pool; s = s + SPU_pool) {
2      LOOP_GamaCore:
3          for (i = 0; i < SPU_pool; i++) {
4              #pragma HLS UNROLL
5
6              gamma[i + s] = elnf[i + s][t] + elnb[i + s][t];
7              normalizer[i + s] = 0.0;
8              for (j = 0; j < hmm.N; j++) {
9                  normalizer[i + s] = ELOGSUM(normalizer[i + s], gamma[j], i);
10             }
11             gamma[i + s] = gamma[i + s] - normalizer[i + s];
12         }
13
14     for (s = 0; s < TDM_steps * SPU_pool; s = s + SPU_pool) {
15         LOOP_XiCore:
16             for (i = 0; i < SPU_pool; i++) {
17                 #pragma HLS UNROLL
18
19                 for (j = 0; j < hmm.N; j++) {
20                     xi[i + s][j] = (hmm.elnA[i + s][j] + elnf[i + s][t]) + (hmm.elnB[j][0[t + 1]] +
21                                     elnb[j][t + 1]);
22                     sumXi = ELOGSUM(sumXi, xi[i + s][j], i);
23                 }
24                 for (j = 0; j < hmm.N; j++) {
25                     xi[i + s][j] = xi[i + s][j] - sumXi;
26                 }
27             }
28     }

```

Jednakże cząstkowe wartości licznika i mianownika muszą być przechowywane i kumulowane oddzielnie dla każdego stanu i dla każdej obserwacji. Oznacza to, że pamięć FPGA musi być w stanie jednocześnie przechowywać zarówno bieżące parametry HMM (A , B , π), jak i nowe (A' , B' , π'). Końcowa reestymacja parametrów odbywa się w bloku algorytmicznym Numerator-Denominator.

7.3.4 Blok Numerator-Denominator D&C

Obliczenia bloku algorytmicznego Numerator-Denominator są zorganizowane podobnie jak w bloku Gamma-Xi. Cząstkowe wartości γ i ξ są akumulowane odpowiednio w liczniku i mianowniku oddzielnie dla każdego stanu i dla każdej obserwacji (patrz Rysunek 7.12). Oznacza to, że pamięć FPGA musi być w stanie jednocześnie przechowywać zarówno bieżące parametry HMM (A , B , π), jak i nowe (A' , B' , π'). Po pełnym zakumulowaniu wartości cząstkowych (po osiągnięciu ostatniej obserwacji w badanej sekwencji) dokonywana jest ostateczna normalizacja. Listing 7.4 przedstawia opis HLS dla etapu określania wartości Numeratora i Denominatora. Zasadniczo wyróżnić można dwie pętle (poddane zrównolegleniu) o etykietach LOOP_NumeDenomCore_A i LOOP_NumeDenomCore_B wykonujące akumulację odpowiednio dla macierzy A' i B' . W dziedzinie logarytmów normalizacja wyniku (uzyskanie rozkładu prawdopodobieństwa) odbywa się poprzez odjęcie zakumulowanego mianownika od licznika dla parametrów macierzy A' i B' . Natomiast prawdopodobieństwo początkowe π' standardowo wynika z wartości γ uzyskanej w chwili czasowej $t = 1$.



Rysunek 7.12: Wizualizacja bloku algorytmicznego przeznaczonego pod obliczenia związane z algorytmem Numerator-Denominator D&C.

(Źródło: opracowanie własne).

Listing 7.4: Opis HLS dla etapu określania wartości Numeratora i Denominatora w celu odtworzenia parametrów modelu w algorytmie Bauma-Welcha.

```

1 if (t < T - 1) {
2   for (s = 0; s < TDM_steps * SPU_pool; s = s + SPU_pool) {
3
4     LOOP_NumeDenomCore_A:
5     for (i = 0; i < SPU_pool; i++) {
6 #pragma HLS UNROLL
7
8       for (j = 0; j < hmm.N; j++) {
9         denominatorA[i + s] = ELOGSUM(denominatorA[i + s], gamma[j], i);
10        numeratorA[i + s][j] = ELOGSUM(numeratorA[i + s][j], xi[i + s][j], i);
11      }
12      updatedHMM.elnA[i + s][j] = numeratorA[i + s][j] - denominatorA[i + s];
13    }
14  }
15 }
16
17 for (s = 0; s < TDM_steps * SPU_pool; s = s + SPU_pool) {
18   for (i = 0; i < SPU_pool; i++) {
19     denominatorB[i + s] = denominatorA[i + s] + gamma[i + s];
20   }
21 }
22
23 
```



```

24
25 for (s = 0; s < TDM_steps * SPU_pool; s = s + SPU_pool) {
26
27     LOOP_NumeDenomCore_B:
28     for (i = 0; i < SPU_pool; i++) {
29 #pragma HLS UNROLL
30
31         for (k = 0; k < hmm.M; k++) {
32             if (O[t] == k) {
33                 numeratorB[i + s][k] = ELOGSUM(numeratorB[i + s][k], gamma[i + s], i);
34             }
35             updatedHMM.elnB[i + s][k] = numeratorB[i + s][k] - denominatorB[i + s];
36         }
37     }
38 }

```

Ponieważ procedura reestymacji wykonana jest w dziedzinie logarytmów to i otrzymane parametry HMM wyrażone są w postaci logarytmów. Taki zapis jest dogodny dla wykonywania kolejnych iteracji obliczeń w logarytmowanej wersji algorytmu Bauma-Welcha, jak i przy dekodowaniu ścieżki stanów w logarytmicznym wariancie algorytmu Viterbiego. Przekształcenie parametrów HMM z dziedziny logarytmicznej do postaci tradycyjnej (np. w celu lepszej interpretacji wartości prawdopodobieństw modelu) umożliwia rozszerzona funkcja wykładnicza $\text{eexp}(x)$.

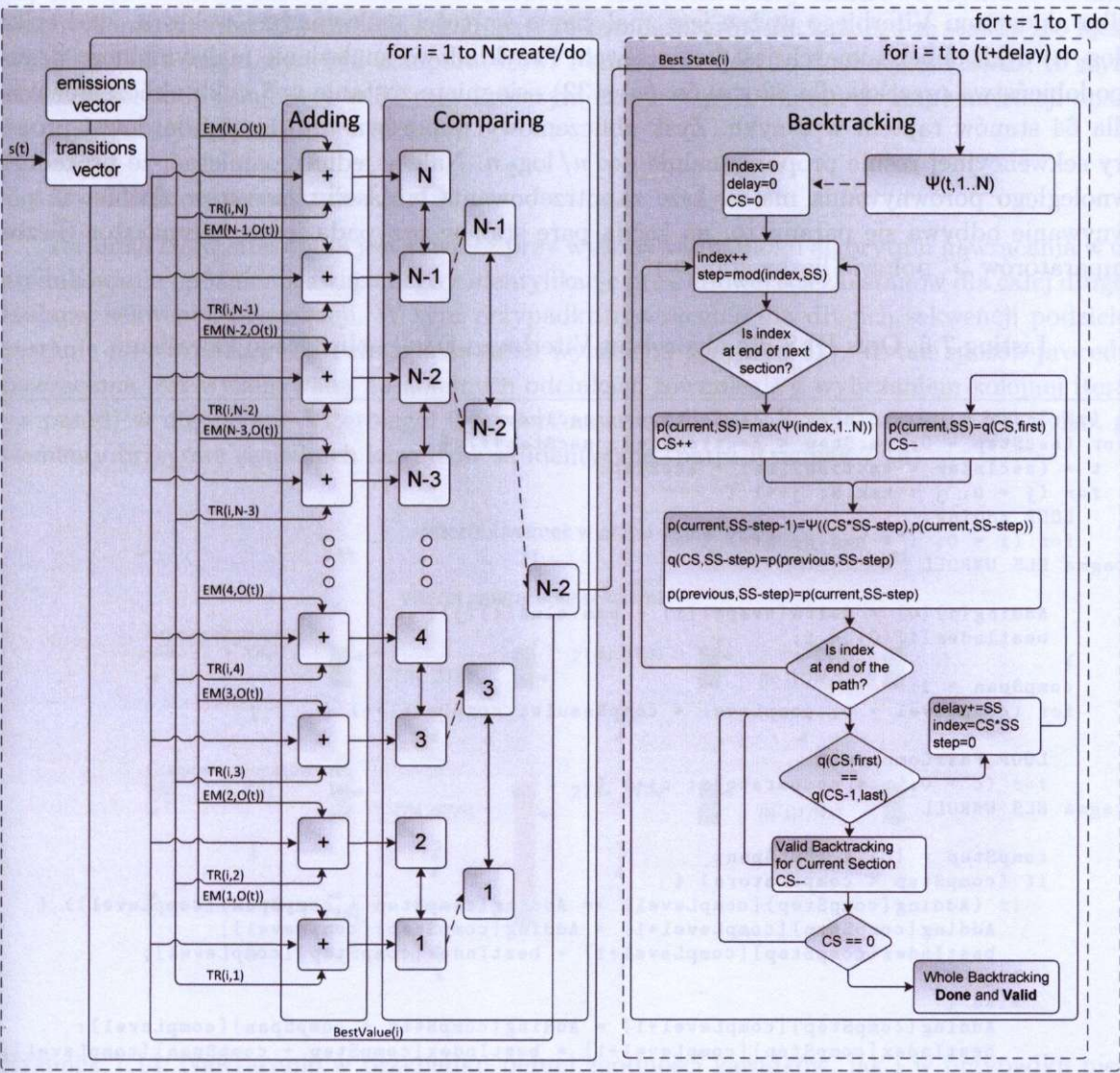
Listing 7.5: Opis HLS dla etapu określania wartości prawdopodobieństwa początkowego dla modelu HMM.

```

1 if(t == 0){
2     for (s = 0; s < TDM_steps * SPU_pool; s = s + SPU_pool) {
3         LOOP_reestimate_Pi:
4         for (i = 0; i < SPU_pool; i++) {
5 #pragma HLS UNROLL
6
7             updatedHMM.elnPi[i + s] = gamma[i + s];
8         }
9     }
10 }

```


7.4 Sprzętowa realizacja algorytmu Viterbi D&C



Rysunek 7.13: Wizualizacja bloku algorytmicznego przeznaczanego pod obliczenia związane z algorytmem Viterbi D&C.

(Źródło: opracowanie własne).

7.4.1 Blok Argmax D&C

W celu uniknięcia problemów numerycznych algorytm Viterbiego również jest obliczany w dziedzinie logarytmów, w której mnożenie prawdopodobieństw przejść, emisji i wartości maksymalnej zastępowane jest operacją sumowania. Sumowanie odbywa się równolegle dla wszystkich stanów. Ostatnim etapem jest znalezienie stanu o najwyższej wartości prawdopodobieństwa przejścia. W tym celu wystarczające jest porównanie wszystkich prawdopodobieństw przejścia, jednakże ta prosta procedura wymaga wykonania n (liczba stanów) porównań sekwencyjnie. Przyspieszenie tego kroku realizowane jest, przez zastosowanie metody: dziel i zwyciężaj. Ogólna idea jest przedstawiona na Rysunek 7.13, gdzie wyróżnione są trzy bloki, a linie przerywane przedstawiają iteracyjne wykorzystanie zasobów. W bloku sumacji „Adding“, wykonywane jest potrójne sumowanie zmiennoprzecinkowa dla wartości maksymalnej (uzyskanej z poprzedniej iteracji) oraz prawdopodobieństwa emisji i prawdopodobieństw przejścia. W bloku porównywania „Comparing“, wszystkie stany są porównywane równolegle parami, co pozwala na wykonanie

procedury w $\log_2 n + 1$ kroków (gdzie n odnosi się do liczby stanów). Podejście tutaj jest analogiczne do algorytmu sortowania merge-sort (Dhawan & Dehon 2015), który wymaga $O(n \log n)$ cykli obliczeniowych, aby uporządkować wszystkie wartości w n -elementowym wektorze. W przypadku algorytmu Viterbiego ważne jest znalezienie wartości maksymalnej wektora, stąd tylko $O(\log_2 n)$ cykli obliczeniowych jest koniecznych. Przykładowo znalezienie maksymalnego prawdopodobieństwa przejścia dla 32 stanów ($n = 32$) osiągnięte zostanie w 5 cykli obliczeniowych, a dla 64 stanów raptem w 6 cykli. Zysk obliczeniowy w porównaniu do standardowej procedury sekwencyjnej rośnie proporcjonalnie do: $n / \log_2 n$. Należy jednak pamiętać, że procedura równoległego porównywania ma większe zapotrzebowania w zasoby sprzętowe. Ponieważ porównywanie odbywa się parami to, na każdą parę stanów przypada jeden komparator (liczba komparatorów $\geq$ połowie liczby stanów).

Listing 7.6: Opis HLS dla algorytmu Viterbiego D&C obliczanego kawałkami.

```

1  for (secIndex = 0; secIndex < allSections; secIndex++) {
2      for (secStep = 0; secStep < sectionSize; secStep++) {
3          t = (secIndex * sectionSize) + secStep;
4          for (j = 0; j < hmm.N; j++) {
5              LOOP_adding:
6                  for (i = 0; i < hmm.N; i++) {
7  #pragma HLS UNROLL
8
9                      Adding[i][0] = delta[swapN][i] + hmm.elnA[i][j];
10                     bestIndex[i][0] = i;
11                 }
12                 compSpan = 1;
13                 for (compLevel = 0; compLevel < compResult; compLevel++) {
14
15                     LOOP_PairCompare:
16                     for (c = 0; c < comparators; c++) {
17  #pragma HLS UNROLL
18
19                         compStep = (c) * compSpan;
20                         if (compStep < comparators) {
21                             if (Adding[compStep][compLevel] >= Adding[compStep + compSpan][compLevel]) {
22                                 Adding[compStep][compLevel+1] = Adding[compStep][compLevel];
23                                 bestIndex[compStep][compLevel+1] = bestIndex[compStep][compLevel];
24                             }
25                             else {
26                                 Adding[compStep][compLevel+1] = Adding[compStep + compSpan][compLevel];
27                                 bestIndex[compStep][compLevel+1] = bestIndex[compStep + compSpan][compLevel];
28                             }
29                         }
30                     }
31                     compSpan *= 2;
32                 }
33                 delta[swapP][j] = Adding[0][compResult] + hmm.elnB[j][0[t]];
34                 psi[t][j] = bestIndex[0][compResult];
35                 if (j == hmm.N - 1) {
36                     if (swapP == 1) {
37                         swapP = 0;
38                         swapN = 1;
39                     }
40                     else {
41                         swapP = 1;
42                         swapN = 0;
43                     }
44                 }
45             }
46         }
47     }

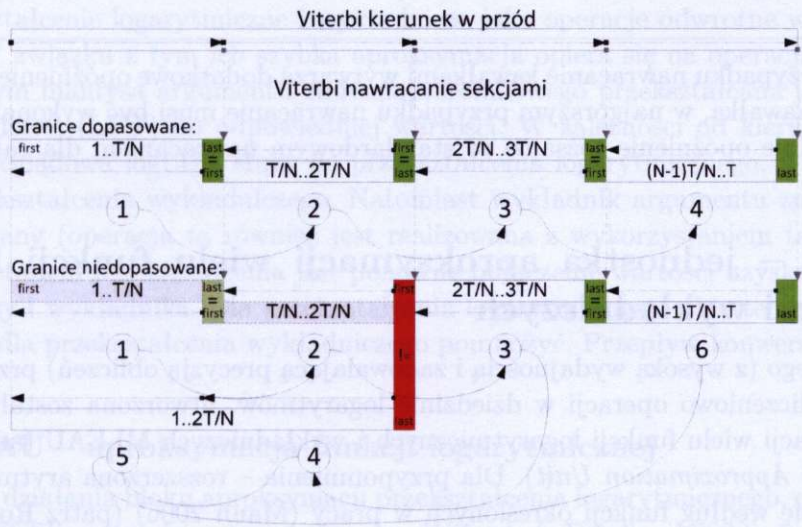
```

Warto podkreślić, że wyniki operacji wykonywanych w bloku dodawania jak i w bloku porównywania przypisane są do danego stanu i mogą być wykonywane dla każdego stanu oddzielnie. Co za tym idzie, możliwa jest implementacja zarówno w pełni równoległa, gdzie dla każdego stanu przypisana jest oddzielna para bloków dodawania i porównywania, jak i implementacja

sekwencyjna, gdzie pojedyncza para bloków wykonuje obliczenia dla wszystkich stanów wykorzystując techniki multipleksacji w dziedzinie czasu. Dla w pełni równoległej implementacji zapotrzebowanie na zasoby wzrasta kwadratowo względem liczby stanów. Przykładowo dla 32 stanów konieczne jest 32 jednostki sumujące i 16 komparatorów dla każdego ze stanów, co łącznie wymaga 1024 sumatory i 512 komparatorów. Pomimo, że jest to dość wysoki koszt, to wyniki obliczeń zostanie osiągnięty w 32 razy krótszym czasie w porównaniu do implementacji opartej na pojedynczej instancji pary bloków.

7.4.2 Blok nawracania D&C

Technika D&C stosowana jest również przy wykonywaniu bloku algorytmu nawracania w celu zredukowania opóźnienia związanego z identyfikacją prawidłowej ścieżki stanów dla całej długości badanej sekwencji obserwacji. W tym przypadku nawracanie dla długich sekwencji podzielone zostanie na równe kawałki (długość kawałka wynosi na przykład 32). W ten sposób procedura nawracania jest wykonywana na kolejnych odcinkach równoległe z wyliczaniem kolejnej iteracji (w przód) w algorytmie Viterbiego. Operacja nawracania kawałkami korzystna jest tylko, gdy elementy brzegowe sąsiadnych kawałków są identyczne (patrz Rysunek 7.14).



Rysunek 7.14: Nawracanie w przypadku pełnej zgodności kawałków, jak i w przypadku niedopasowania granic kawałków.
(Źródło: opracowanie własne).

Kawałki skonstruowane są tak, że ostatni element poprzedniego kawałka pokrywa się z pierwszym elementem kawałka bieżącego. Podczas nawracania dla bieżącego kawałka przeprowadzany jest test zgodności polegający na porównaniu jego pierwszego elementu z ostatnim elementem kawałka poprzedniego. Test zgodności oparty jest na zasadzie, że skoro poszukiwany jest stan o największym prawdopodobieństwie przejścia, to może być tylko jeden taki stan. Tak więc, jeżeli bieżący kawałek prowadzi właśnie do tego samego stanu, na którym kończy się kawałek poprzedni, to odkryta dla tych dwóch kawałków najbardziej prawdopodobna ścieżka stanów jest wiarygodna. Jeżeli ta sprzyjająca sytuacja powtarza się wzdłuż całej długości badanej sekwencji, to dzięki zrównolegleniu nawracania (względem indukcji algorytmu Viterbiego) wprowadzone opóźnienie ograniczone jest do długości pojedynczego kawałka. Jeżeli natomiast elementy brzegowe dwóch sąsiadnych kawałków różnią się od siebie, to nawracanie musi być powtórzone kolejno dla każdego poprzedniego kawałka aż do kawałka, dla którego elementy brzegowe ponownie są identyczne (lub do początku sekwencji).

Listing 7.7: Opis HLS dla algorytmu nawracania kawałkami.

```

1  for (secIndex = 0; secIndex < allSections; secIndex++) {
2      mostLikelyStates[t] = bestIndex[0][compResult];
3
4      LOOP_MLS1:
5      for (s = t - 1; s > sectionBegin; s--) {
6          #pragma HLS PIPELINE
7
8          mostLikelyStates[s] = psi[s + 1][mostLikelyStates[s + 1]];
9      }
10     if (secIndex > 0) {
11         if (mostLikelyStates[sectionBegin - 1] != psi[sectionBegin][mostLikelyStates[
12             sectionBegin]]) {
13             LOOP_MLS2:
14             for (s = t - 1; s >= sectionBegin - sectionSize; s--) {
15                 #pragma HLS PIPELINE
16
17                 mostLikelyStates[s] = psi[s + 1][mostLikelyStates[s + 1]];
18             }
19             errSections++;
20         }
21         sectionBegin = t + 1;
22     }

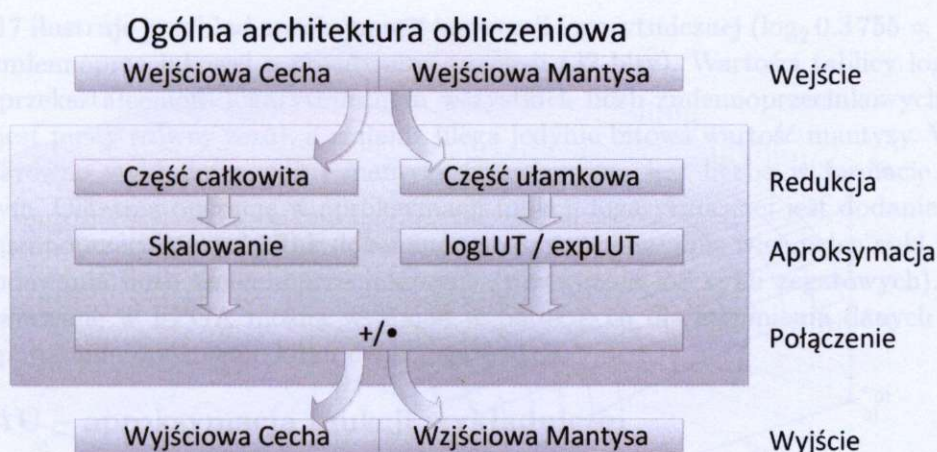
```

W najlepszym przypadku nawracanie kawałkami wytwarza dodatkowe opóźnienie równe długości pojedynczego kawałka, w najgorszym przypadku nawracanie musi być wykonane dla całej sekwencji, co powoduje opóźnienie tożsame ze standardowym nawracaniem dla całej długości badanej sekwencji.

7.5 MLEAU – jednostka aproksymacji wielu funkcji logarytmicznych i wykładniczych

W celu skutecznego (z wysoką wydajnością i zadowalającą precyzją obliczeń) przeprowadzenia kosztownych obliczeniowo operacji w dziedzinie logarytmów, utworzona została specjalna jednostka aproksymacji wielu funkcji logarytmicznych i wykładniczych MLEAU (ang. *Multiple Logarithm Exponent Approximation Unit*). Dla przypomnienia – rozszerzona arytmetyka logarytmiczna odbywa się według funkcji określonych w pracy (Mann 2006) (patrz Rozdział 2.2). W ukrytych modelach Markowa obliczenia wykonywane są na macierzach przejść i emisji. Zauważając, że w macierzach prawdopodobieństwa wyrażone są w zakresie od 0 do 1, a co za tym idzie, logarytm z wartości zawartych w tym przedziale jest liczbą ujemną. Należy zauważyć w tym kontekście, że o ile algorytm Viterbiego można dość łatwo wyrazić w dziedzinie logarytmów (w kroku indukcyjnym wykonywany jest logarytm iloczynu, co można wyrazić sumą logarytmów), to reprezentacja algorytmu Forward-Backward w dziedzinie logarytmów nie jest tak oczywista, ponieważ tu krok indukcyjny wymaga operacji sumy, a zatem przekłada się to na logarytm sumy (dla którego nie ma wygodnej bezpośredniej tożsamości matematycznej). Stąd, w powyższym zestawie funkcji, szczególnie interesująca jest procedura, która wymaga zastosowania przekształcenia wykładniczego, po czym przekształcenia logarytmicznego (patrz wzory 2.14 i 2.15).

Dla sprawnego przeprowadzenia tego typu obliczeń proponowana jest jednostka MLEAU stanowiąca rozszerzenie algorytmu ICSlog (Vinyals & Friedland 2008) przez dodanie funkcjonalności przekształcenia wykładniczego oraz interfejs równoległego dostępu, co przekłada się na zysk w wydajności.

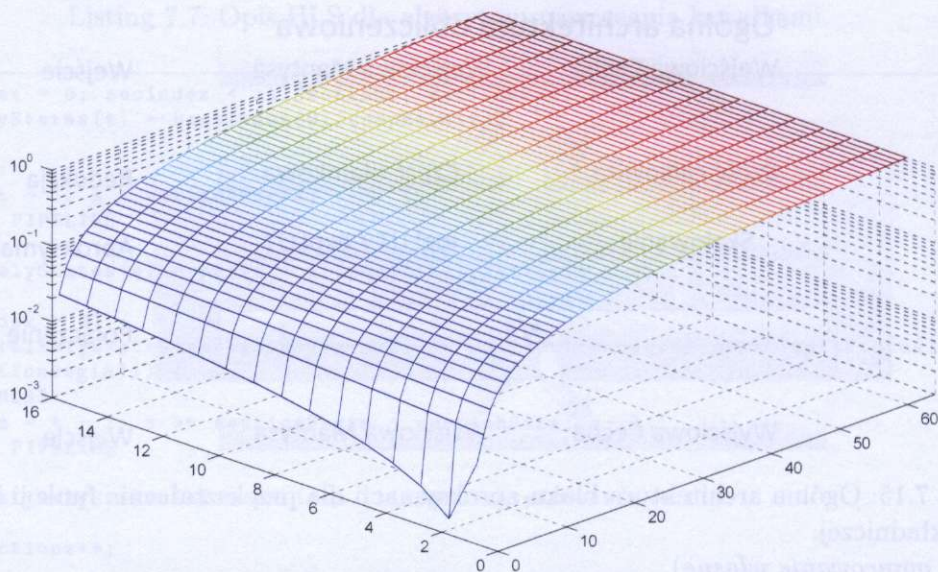


Rysunek 7.15: Ogólna architektura bloku aproksymacji dla przekształcenia funkcji logarytmicznej i wykładniczej.
(Źródło: opracowanie własne).

Przekształcenie logarytmiczne i wykładnicze jako operacje odwrotne wymagają podobnych działań. W związku z tym ich szybka aproksymacja opiera się na operacji indeksowania tablicy, w którym mantysa argumentu zmiennoprzecinkowego przekształcana jest za pomocą tablicy przeglądowej (LUT) do odpowiedniej wartości. W zależności od kierunku przekształcenia, tablica przeglądowa logLUT służy do przekształcenia logarytmicznego, a tablica expLUT służy do przekształcenia wykładniczego. Natomiast wykładnik argumentu zmiennoprzecinkowego jest skalowany (operacja ta również jest realizowana z wykorzystaniem tablicy przeglądowej). Ostatnim etapem przekształcenia jest ponowne połączenie wartości uzyskanych z przekształcenia mantysy i wykładnika. Dla przekształcenia logarytmicznego obie te wartości należy dodać, natomiast dla przekształcenia wykładniczego pomnożyć. Przepływ konwersji pokazuje Rysunek 7.15.

7.5.1 LAU – aproksymacja funkcji logarytmicznej

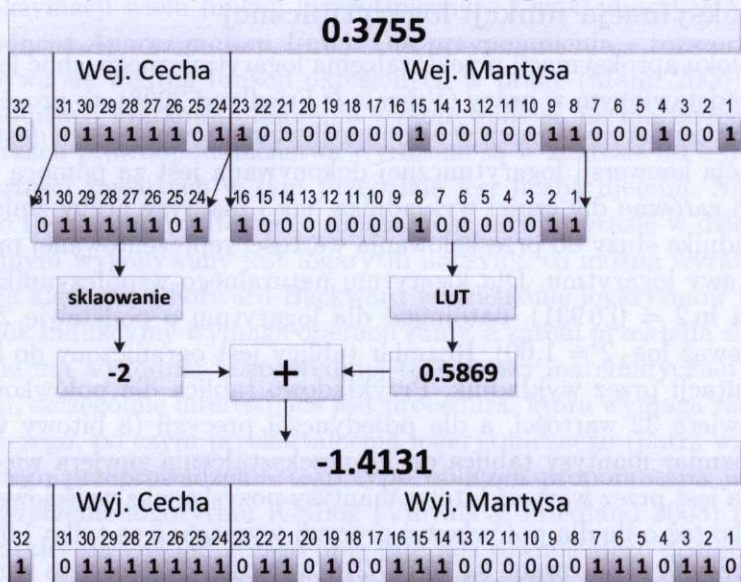
Zasada działania bloku aproksymacji przekształcenia logarytmicznego, choć jest zbieżna z algorytmem ICSIllog przedstawionym w pracy (Vinyals & Friedland 2008), to proces dokonywania przekształcenia poddany został rearanżacji w celu umożliwienia równoległego dostępu. Dla przypomnienia aproksymacja konwersji logarytmicznej dokonywana jest za pomocą tablic przeglądowych dedykowanych zarówno dla części wykładnika, jak i mantysy liczby zmiennoprzecinkowej. Tablica dla wykładnika służy do przeskalowania wartości reprezentowanej przez wykładnik w zależności od podstawy logarytmu. Dla logarytmu naturalnego współczynnik skalujący wynosi 0.6931 (ponieważ $\ln 2 = 0.6931$), natomiast dla logarytmu o podstawie 2 współczynnik ten wynosi 1.00 (ponieważ $\log_2 2 = 1.00$). Rozmiar tablicy jest ograniczony do liczby wartości możliwych do reprezentacji przez wykładnik. Przykładowo tablica dla połówkowej precyzji (5 bitowy wykładnik) zawiera 32 wartości, a dla pojedynczej precyzji (8 bitowy wykładnik) już 256. Ze względu na rozmiar mantysy tablica do jej przekształcenia zawiera więcej elementów. Tablica ta indeksowana jest przez wartość bitową mantysy pozyskanej z wejściowej liczby zmiennoprzecinkowej. Dlatego też o ile dla połówkowej precyzji wymagane są tylko 1024 wartości, to już dla pojedynczej precyzji tablicowane są 16 777 216 wartości. Wartości te zorganizowane są z wykorzystaniem wielu tablic miniLUT.



Rysunek 7.16: Wizualizacja zawartości tablic przeglądowych miniLUT dedykowych dla mantysy w przekształceniu logarytmicznym o podstawie 2 (16 tablic po 64 wartości).
(Źródło: opracowanie własne).

Rysunek 7.16 przedstawia mapowanie 1024 wartości (dla połówkowej precyzji) na 16 tablic miniLUT. Dla uwypuklenia różnic pionową oś wartości została przedstawiona w dziedzinie logarytmicznej. Tablice wybierane są przez kombinacje 4 najmniej znaczących bitów mantysy z zadanej wartości wejściowej (np.: "0010"_{BIN} = miniLUT(2), "1110"_{BIN} = miniLUT(14)). Każda tablica zawiera 64 wyniki przekształcenia odpowiadające wartościom wejściowym o ustalonej końcówce bitowej. Wybrana tablice miniLUT indeksowana jest pozostałą 6 bitową częścią mantysy.

Przykład aproksymacji funkcji logarytmicznej



Rysunek 7.17: Przykład operacji aproksymacji przekształcenia dla funkcji logarytmicznej o podstawie 2.
(Źródło: opracowanie własne).

Możliwe jest również wykorzystanie tylko pewnej najbardziej znaczącej (zaczynając od bitu MSB) części mantysy do przekształcenia, wtedy jednak następuje odpowiednia redukcja precyzji.

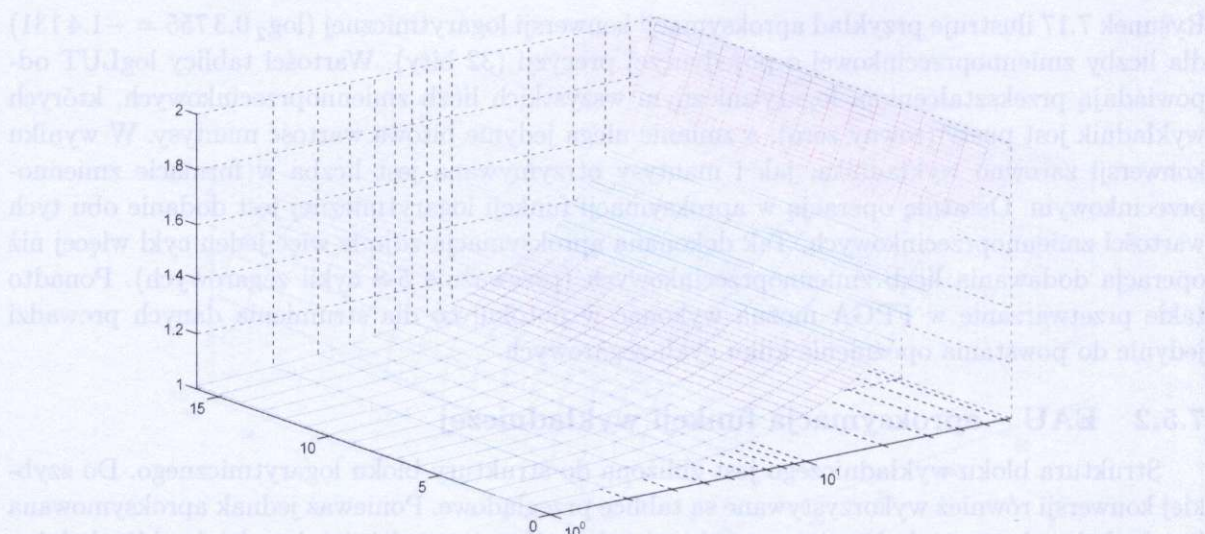
Rysunek 7.17 ilustruje przykład aproksymacji konwersji logarytmicznej ($\log_2 0.3755 = -1.4131$) dla liczby zmiennoprzecinkowej o pojedynczej precyzji (32 bity). Wartości tablicy logLUT odpowiadają przekształceniom logarytmicznym wszystkich liczb zmiennoprzecinkowych, których wykładnik jest pusty (równy zero), a zmianie ulega jedynie bitowa wartość mantysy. W wyniku konwersji zarówno wykładnika, jak i mantysy otrzymywana jest liczba w formacie zmiennoprzecinkowym. Ostatnią operacją w aproksymacji funkcji logarytmicznej jest dodanie obu tych wartości zmiennoprzecinkowych. Tak dokonana aproksymacja zajmie więc jeden cykl więcej niż operacja dodawania liczb zmiennoprzecinkowych (przeważnie 5-8 cykli zegarowych). Ponadto takie przetwarzanie w FPGA można wykonać w potoku, co dla strumienia danych prowadzi jedynie do powstania opóźnienia kilku cykli zegarowych.

7.5.2 EAU – aproksymacja funkcji wykładniczej

Struktura bloku wykładniczego jest zbliżona do struktury bloku logarytmicznego. Do szybkiej konwersji również wykorzystywane są tablice przeglądowe. Ponieważ jednak aproksymowana jest funkcja odwrotna do logarytmowania, to też zarówno wartości, jak i podział tablic dedykowany jest dla przekształcenia wykładniczego. O ile dla przekształcenia logarytmicznego dokonywany jest stały podział na wykładnik i mantysę, które są przetwarzane przez oddzielne tablice przeglądowe, to dla przekształcenia wykładniczego wartość wejściową należy podzielić na część całkowitą i część ułamkową. Jednakże w formacie zmiennoprzecinkowym bitowa reprezentacja części całkowitej i części ułamkowej jest ruchoma i zależy od interpretowanej liczby (wyrażonej wykładnikiem i mantysą), a dokładnie od wartości wykładnika. Dlatego też w pierwszym kroku należy ustalić, która część ciągu bitów określa liczbę całkowitą (pozostała część ciągu bitów określa więc część ułamkową). Teoretycznie miejsce przecinkowe znajdować się może w dowolnej części bitowego formatu liczby zmiennoprzecinkowej. Korzystając jednak z ograniczenia precyzji odwzorowania liczb dla danego formatu, pole poszukiwań zawęzić można do liczb, których wynik przekształcenia wykładniczego daje się jeszcze zapisać na danym formacie zmiennoprzecinkowym. Dla przypomnienia w zależności od precyzji formatu dla funkcji wykładniczej o podstawie 2 możliwy zakres liczbowy argumentu to:

- od -24 do 15 dla połówkowej precyzji,
- od -150 do 127 dla pojedynczej precyzji,
- od -1075 do 1023 dla podwójnej precyzji.

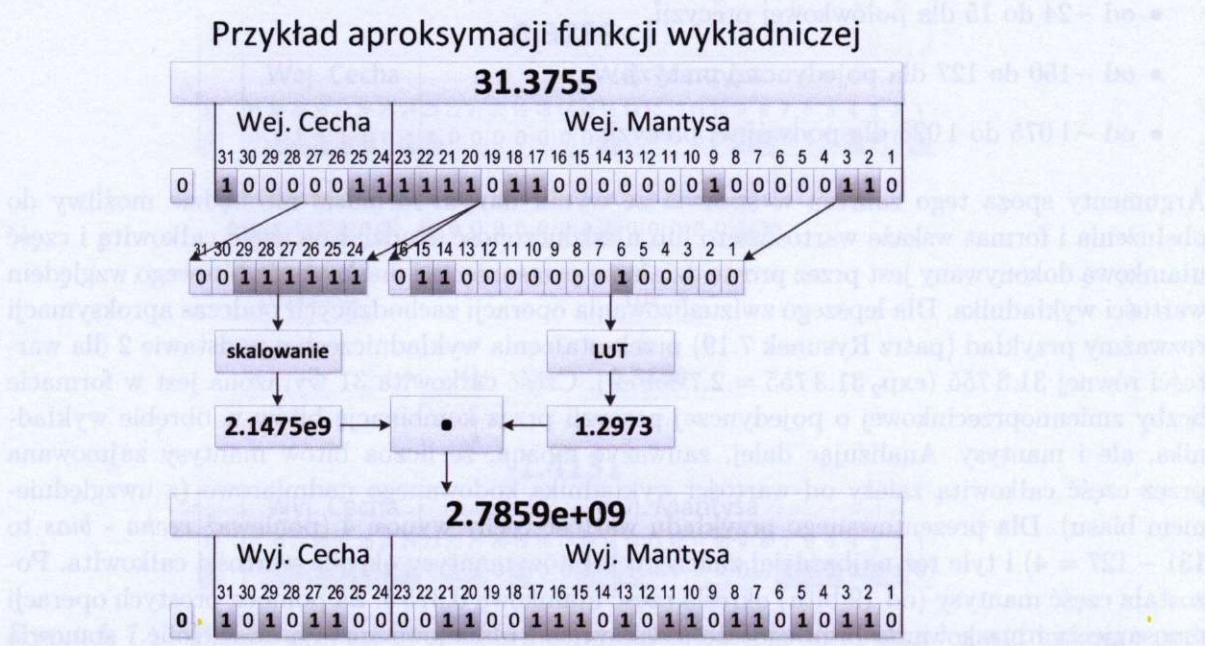
Argumenty spoza tego zakresu w sprawie ze wynikiem danym formacie nie będzie możliwy do obsłużenia i format wskaże wartość zero lub nieskończoność. Podział na część całkowitą i część ułamkową dokonywany jest przez proste działanie przesunięcia i maskowania bitowego względem wartości wykładnika. Dla lepszego zwizualizowania operacji zachodzących podczas aproksymacji rozważmy przykład (patrz Rysunek 7.19) przekształcenia wykładniczego o podstawie 2 dla wartości równej 31.3755 ($\exp_2 31.3755 = 2.7859e9$). Część całkowita 31 wyrażona jest w formacie liczby zmiennoprzecinkowej o pojedynczej precyzji przez kombinację bitów w obrębie wykładnika, ale i mantysy. Analizując dalej, zauważyć można, że liczba bitów mantysy zajmowana przez część całkowitą zależy od wartości wykładnika kodowanego nadmiarowo (z uwzględnieniem biasu). Dla prezentowanego przykładu wartość cechy wynosi 4 (ponieważ $cecha - bias$ to $131 - 127 = 4$) i tyle też najbardziej znaczących bitów mantysy określa wartość całkowita. Pozostała część mantysy (od 19 bitu) określa część ułamkową 0.3755. Za pomocą prostych operacji przesunięcia i maskowania bitowego część całkowita i ułamkowa zostają oddzielone i stanowią wejście dla odpowiedniej tablicy przeglądowej. Obie tablice przechowują wyniki funkcji wykładniczej, z tą różnicą, że tablica dla wartości całkowitych integerLUT przetrzymuje wyniki dla liczb całkowitych od -126 do 127, natomiast tablica dla wartości ułamkowych expLUT przetrzymuje wyniki funkcji wykładniczej dla liczb w zakresie od 1.00 do 2.00 z rozdzielczością odpowiednią dla danej długości mantysy.



Rysunek 7.18: Wizualizacja zawartości tablic przeglądowych miniLUT dedykowanych dla mantysy w przekształceniu wykładniczym o podstawie 2 (16 tablic po 64 wartości).
(Źródło: opracowanie własne).

Rysunek 7.18 przedstawia mapowanie 1 024 wartości (dla połówkowej precyzji) na 16 tablic miniLUT. Analogicznie jak dla funkcji logarytmicznej również tu tablice wybierane są przez kombinacje 4 najmniej znaczących bitów w wartości wejściowej, a pozostała część bitowa mantysy stanowi indeks tablicy.

Ostatnią operacją w aproksymacji funkcji wykładniczej jest przemnożenie obu otrzymanych wartości zmiennoprzecinkowych. Tak dokonana aproksymacja zajmie więc jeden cykl więcej niż operacja mnożenia liczb zmiennoprzecinkowych (przeważnie 5-8 cykli zegarowych). Również tutaj przetwarzanie w FPGA wykonać można w potoku, co wprowadzi jedynie do powstania opóźnienie kilku cykli zegarowych.



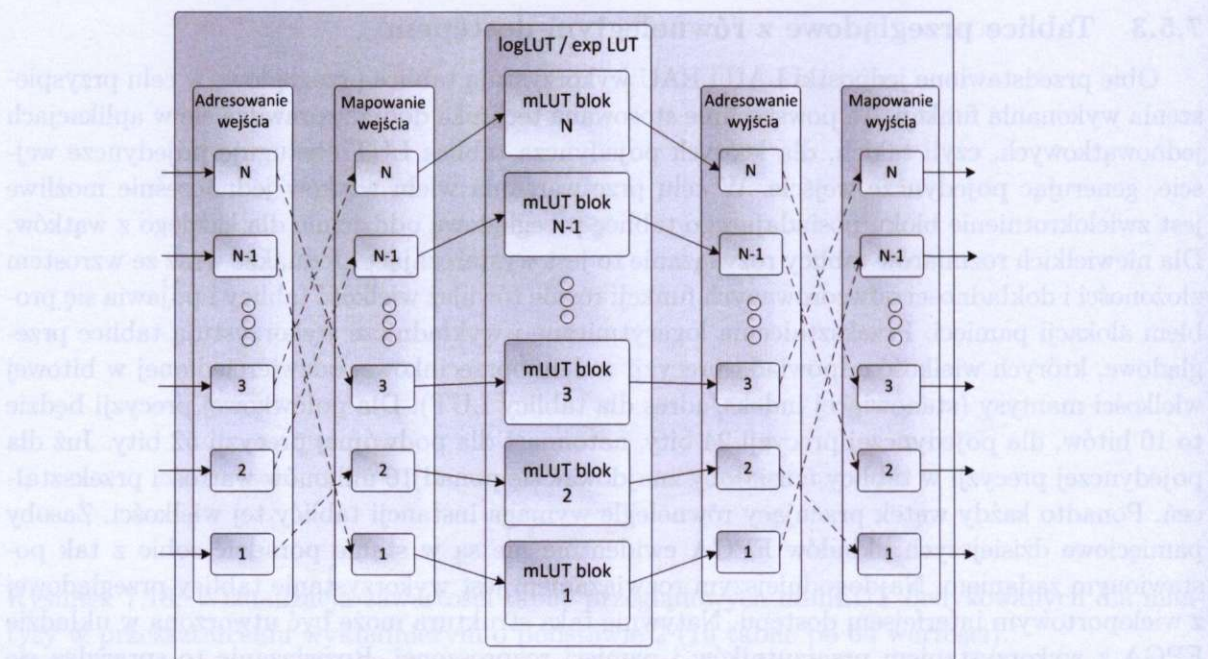
Rysunek 7.19: Przykład operacji aproksymacji przekształcenia dla funkcji wykładniczej o podstawie 2.
(Źródło: opracowanie własne).

7.5.3 Tablice przeglądowe z równoległym dostępem

Obie przedstawione jednostki LAU i EAU wykorzystują tablice przeglądowe w celu przyspieszenia wykonania funkcji. Ta powszechnie stosowana technika dobrze sprawdza się w aplikacjach jednowątkowych, czyli takich, dla których pojedyncza tablica LUT obsługuje pojedyncze wejście, generując pojedyncze wyjście. W celu przetwarzania wielu wątków jednocześnie możliwe jest zwielokrotnienie bloku posiadającego tablicę przeglądową oddzielnie dla każdego z wątków. Dla niewielkich rozmiarów tablicy rozwiązanie to jest wystarczające. Jednakże wraz ze wzrostem złożoności i dokładności odwzorowanych funkcji rośnie również wielkość tablicy i pojawia się problem alokacji pamięci. Przekształcenia logarytmiczne i wykładnicze wykorzystują tablice przeglądowe, których wielkość odpowiada precyzji zmiennoprzecinkowej odzwierciedlonej w bitowej wielkości mantysy (stanowiącej indeks/adres dla tablicy LUT). Dla połówkowej precyzji będzie to 10 bitów, dla pojedynczej precyzji 24 bity, natomiast dla podwójnej precyzji 52 bity. Już dla pojedynczej precyzji w tablicy musiałoby znajdować się ponad 16 milionów wartości przekształceń. Ponadto każdy wątek pracujący równolegle wymaga instancji tablicy tej wielkości. Zasoby pamięciowe dzisiejszych układów FPGA ewidentnie nie są w stanie poradzić sobie z tak postawionym zadaniem. Najdogodniejszym rozwiązaniem jest wykorzystanie tablicy przeglądowej z wieloportowym interfejsem dostępu. Natywnie taka struktura może być utworzona w układzie FPGA z wykorzystaniem przerzutników i pamięci rozproszonej. Rozwiązanie to sprawdza się dla struktur okupujących niewielki obszar logiki i pamięci. Przykładowo implementacja w pełni równoległej tablicy przeglądowej zawierającej 16 elementów. Realizacja przekształcenia funkcji o wysokim stopniu złożoności (1024 i więcej elementów) w oparciu o pamięć rozproszoną bardzo często doprowadza do problemów czasowych i utrudnia operacje Place&Route, co ma negatywny wpływ na wydajność implementacji.

Do alokacji większych obszarów pamięci w FPGA przeznaczone są prymitywy pamięci blokowych BRAM. Standardowo zasoby BRAM zapewniają jedynie pojedynczy lub podwójny port dostępu. Potrzebna jest więc struktura kombinatoryczna, która pozwoli na zagospodarowanie bloków pamięci tak, aby możliwy był równoległy wieloportowy dostęp do wybranych zasobów. Taki podział zasobów wpisuje się w ramy techniki „dziel i zwyciężaj” D&C. Dlatego też w miejsce jednej obszernej tablicy przeglądowej implementowane jest wiele mniejszych LUT (zwanych dlatego miniLUT). Każda miniLUT obejmuje inny obszar funkcyjny, co w połączeniu odwzorowuje kompletną funkcję przekształcenia. Rysunek 7.20 przedstawia ogólną ideę podziału. Wybór poszczególnych tablic miniLUT dokonywany jest w zależności od wartości wejściowej. Stąd, zasada wyboru tablicy jest podobna do pamięci adresowanej treścią (ang. *CAM – Content Addressable Memory*) (Soni & Dakhole 2014) z tą różnicą, że adresowanie tablic opiera się na mniej znaczącej części bitowej (LSB) wartości wejścia.

Chociaż lokalnie każda miniLUT obsługuje tylko pojedynczy dostęp (tablica utworzona jest na bloku pamięci BRAM z pojedynczym portem), to globalnie dostęp do funkcji przekształcenia realizowany jest równolegle. Emulacja pełnej równoległości przetwarzania dokonywana jest poprzez wykorzystanie mapowania blisko asocjacyjnego dla wejścia i wyjścia (Dhawan & DeHon 2013). Dlatego też cały moduł umożliwia dostęp pamięci o stopniu zrównoleglenia nie większym niż liczba instancji miniLUT, a ponadto wprowadza pewne ograniczenia co do optymalnej struktury przetwarzanych danych wejściowych. Jeśli wszystkie wartości wejściowe mają inną część LSB (przeznaczona do wyboru tablicy miniLUT zgodnie z zasadami adresowania treścią), to przekształcenie dokonywane jest jednocześnie dla wszystkich wartości wejściowych. Jednakże, jeżeli zostanie rozpoznany konflikt dostępu, gdzie część LSB jest taka sama dla kilku wartości wejściowych (tym samym wybrana jest ta sama tablica miniLUT), to operacja zostanie wykonana w kolejce, powodując przestój i opóźnienie w konwersji. Tego typu konflikty rozwiązywane są przez tablice kodów stopowych, dzięki której dokonywana jest reorganizacja wejścia tablic. Wydajność zrównoleglenia zależy więc od zróżnicowania wartości wejściowych.



Rysunek 7.20: Blisko asocjacyjne mapowanie wejścia i wyjścia dla zrównoleglenia konwersji LUT. (Źródło: opracowanie własne).

Dla przekształcenia wartości logarytmicznej dokonywanej dla algorytmu Forward-Backward odnotowany całkowity spadek wydajności był na poziomie 33% bez względu na stopień zrównoleglenia. Oznacza to, że przy wykorzystaniu 16 tablic miniLUT stopień realnego zrównoleglenia wyniósł około 11, analogicznie dla 8 tablic miniLUT stopień realnego zrównoleglenia wyniósł około 5. Ze względu na blisko asocjacyjne mapowania dla wejścia i dla wyjścia, zapotrzebowanie na zasoby FPGA wzrasta kwadratowo do liczby portów. Jednakże, biorąc pod uwagę korzyści z równoległości dostępu (oszczędności w wykorzystaniu pamięci), zwiększone wykorzystania zasobów jest w pełni akceptowalne.

Tabela kodów stopowych

Kody stopowe wykorzystywane są w celu unikania konfliktów. Jeżeli dwie lub więcej wartości wejściowych próbuje jednocześnie zaadresować ten sam obszar (skorzystać z tego samego zasobu), to za pomocą kodów stopowych dostęp do zasobu zostanie kolejgowany tak, aby możliwe było sekwencyjne obsłużenie żądania.

Rozpatrzmy 16 żądań uzyskania dostępu do zasobu. Tabela 7.2 przedstawia przykład, gdzie wartości wejściowe wymagają dostępu do wybranych zasobów.

Wejście	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Żądany zasób	2	4	4	6	7	11	12	4	11	5	7	15	8	7	7	4

Tabela 7.2: Przykład żądań uzyskania dostępu do wybranych zasobów. (Źródło: opracowanie własne).

W przedstawionym przykładzie dostęp do części zasobów żądany jest jednocześnie przez wiele wartości wejściowych. Liczbę żądań dla każdego zasobu określić można algorytmicznie, dokonując kolejgowanie żądań uzyskania dostępu do zasobu względem każdego zasobu. Wynik można zwizualizować w postaci mapy dostępu, co przedstawia Tabela 7.3. Ostatnia kolumna określa sumarycznie, ile żądań dostępu przypada na dany zasób. Określanie mapy dostępu można wykonać równolegle dla każdego zasobu. Niemniej jednak nadal algorytm ten posiada kwadratową złożoność przestrzenną. W układzie FPGA zrównoleglenie takiej pętli oznacza więc kwadratowe zapotrzebowanie sprzętowe (tj. jednoczesne porównanie 16 wejść wymaga 256 komparatorów).

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	Wynik
1																	0
2	x																1
3																	0
4		x	x					x								x	4
5										x							1
6				x													1
7					x						x			x	x		4
8													x				1
9																	0
10																	0
11						x			x								2
12							x										1
13																	0
14																	0
15												x					1
16																	0

Tabela 7.3: Mapa ądań dostępu do zasobów względm wartości wejściowej.
(Źródło: opracowanie własne).

Ponieważ jednak w celu skutecznego wykorzystania zasobów bardziej istotne jest jak kolejno obsłużyć konflikty. Dlatego też zamiast mapy dostępu wyliczana jest mapa konfliktów (patrz Tabela 7.4), gdzie iteracja dokonywana jest tylko dla niesprawdzonych wariantów wejściowych (patrz Listing 7.8). W ten sposób złożoność przestrzenna maleje o połowę, tym samym liczba wymaganych komparatorów w FPGA również jest odpowiednio mniejsza. Mapa konfliktów nie definiuje dokładnie, dla którego wejścia ądany jest który zasób, a raczej wystąpienie konfliktu dla wejścia względm zasobu lub dla zasobu względm wejścia.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1																
2	x															
3																
4		x	x													
5																
6				x												
7					x											
8				x												
9																
10					x											
11						x	x		x							
12							x									
13								x								
14							x									
15												x				
16				x												

	1	1	1	3	2	1	3	1	1	0	0	1	0	0	0	
--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--

Tabela 7.4: Mapa konfliktów dostępu do zasobów względm wartości wejściowej.
(Źródło: opracowanie własne).

Mapa konfliktu wyliczana jest równolegle w każdym cyklu zegarowym i na jej podstawie tworzona jest tabela kodów stopowych, która określa blokowanie wejścia. Tabela 7.5 przedstawia wyliczone kody stopowe dla prezentowanego przykładu. Jak widać, w ciągu czterech cykli zegarowych wszystkie wejścia uzyskały dostęp do zasobów. ądanie dostępu przez wejście oznakowane kodem stopu jest blokowane, w wyniku czego inne wejście (będące w konflikcie) otrzymuje do-

step. Żądanie wejście obsługowanego w danym cyklu automatycznie wygasa i nie bierze udziału w kolejnych cyklach. Jeżeli tabela kodów stopowych wypełniona jest zerami, to oznacza to, że wszystkie wejścia zostały obsługowane.

Numer wejścia:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Wykryty konflikt:	2	4	4	6	7	11	12	4	11	5	7	15	8	7	7	4

Numer wejścia:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Kod stopowy w I iteracji:	0	1	1	0	1	0	0	1	0	0	1	0	0	1	0	0
Kod stopowy w II iteracji:	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0
Kod stopowy w III iteracji:	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Kod stopowy w IV iteracji:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabela 7.5: Kody stopowe określające blokowane wejścia w danej iteracji.
(Źródło: *opracowanie własne*).

Listing 7.8: Sprzętowe (opisane językiem VHDL) wykrywanie konfliktów dostępu i określanie kodów stopowych.

```

1 stopWord:= (others => '0');
2 for indexa in 0 to instances loop
3   for indexb in 1 + indexa to instances loop
4     if (( '0' & InputMap(indexa)(3 downto 0)) = ( stop(indexa) & InputMap(indexb)(3 downto
5       stopCode(indexa)(indexb) := '1';
6       stopWord(indexb)         := '1';
7     else
8       stopCode(indexa)(indexb) := '0';
9     end if;
10  end loop;
11 end loop;
12
13 for indexb in instances downto 0 loop
14   if(stopCode(indexb) /= x"0000")then
15     for shift in indexb + 1 to instances loop
16       if((stopCode(indexb) and stopCode(shift)) /= x"0000")then
17         stop(shift) := '0';
18       end if;
19     end loop;
20     stop(indexb) := '1';
21   end if;
22 end loop;
23 stopOut  <= stop;
```

7.6 Wyniki implementacji

Vivado HLS jako narzędzie do tworzenia sprzętowych realizacji algorytmów przetwarzania sygnałów i danych pozwala używać wyrażeń matematycznych i konstrukcji sterujących języków programowania typu C/C++. Podczas projektowania w stylu RTL pierwszą barierę psychologiczną stanowi zwykle potrzeba zrozumienia, że operatory HDL nie są realizowane w kolejności pojawienia się, zaś jednocześnie. W HLS właśnie ta cecha projektowania dla FPGA została wyeliminowana. Jednakże aplikacja HLS sama w sobie nie gwarantuje wzrostu wydajności. Nie każdy program napisany w języku wysokiego poziomu, może być skutecznie zrealizowany w FPGA. Tak więc dla systemów przetwarzania sygnałów i danych korzystnie jest wykonanie architektury algorytmiczno-obliczeniowej w HLS, która może być suplementowana wyspecjalizowanymi rdzeniami IP opracowanymi w języku opisu sprzętu.

Wszystkie prezentowane moduły zostały zweryfikowane poprzez symulacje behawioralne w środowisku Vivado HLS, gdzie celem testów było porównanie wyników uzyskanych za pomocą algorytmów skompilowanych do RTL z wynikami równoważnych funkcji napisanych w języku wysokiego poziomu jak C/C++. Dodatkowo, MLEAU jako wyspecjalizowana jednostka aproksymacji przekształcenia logarytmicznego i wykładniczego poddana została również osobnej ocenie.

7.7 Synteza i wykorzystanie zasobów FPGA

Projektowanie architektury obliczeniowej na wysokim poziomie abstrakcji skutkuje opisem RTL silnie zależnym od zastosowanych dyrektyw HLS. Możliwe jest uzyskanie oszczędnej struktury, która wykonuje obliczenia bardziej sekwencyjnie przy użyciu technik multipleksowania lub bardziej wymagającej struktury o wyższym stopniu równoległości. W proponowanym rozwiązaniu stopień równoległości ustalany jest względem potęgi dwójki w zależności od dostępnych zasobów i wielkości modelu (np. dla HMM z 30 stanami stopień zrównoleglenia może wynosić 1, 2, 4, 8, 16 lub w najlepszym przypadku 32). W związku z tym w tabeli 7.6 przedstawiono streszczenie wykorzystania zasobów bloków algorytmów Bauma-Welcha, Viterbi zarówno dla wariantu o stopniu zrównoleglenia 1, 8, 16 jak i 32.

Stopień zrównoleglenia	1	8	16	32
Blok Viterbi				
BRAM_18K	64	64	64	64
DSP48E	2	16	34	74
Przerzutniki	6 524	12 167	15 246	21 826
LUT	3 328	14 369	20 922	36 890
Blok Baum-Welch				
BRAM_18K	640	640	640	640
DSP48E	10	22	38	70
Przerzutniki	5 424	21 635	40 488	77 971
LUT	7 666	45 898	90 366	179 659
Blok MLEAU				
BRAM_18K	256	256	256	256
DSP48E	3	24	48	96
Przerzutniki	498	2 351	4 524	8 644
LUT	359	2 985	7 581	22 330

Tabela 7.6: Wykorzystanie zasobów FPGA w zależności od stopnia zrównoleglenia. (Źródło: *opracowanie własne*).

Wygenerowany projekt ma wysoką ocenę częstotliwości taktowania. Wynika to głównie z faktu, że w tekście źródłowym znajduje się funkcja, odwołująca się do gotowego rdzenia IP wykonującego szybką aproksymację przekształcenia logarytmicznego. Ze względu na kombinatoryczny charakter rozwiązywania konfliktów żądań w MLEAU, zapotrzebowanie na zasoby sprzętowe rośnie kwadratowo do stopnia zrównoleglenia. Dlatego też, aby utrzymać zasoby na rozsądnym poziomie, stopień zrównoleglenia dla pojedynczej instancji MLEAU powinien wynosić 8 lub 16. Natomiast liczba instancji ograniczona jest jedynie rozmiarem FPGA.

Interwałowy charakter opóźnienia związany jest z tym, że komponenty projektu są generowane przez HLS za pomocą mechanizmu łańcucha wymiany informacji. Przeglądając kod wynikowy VHDL można zauważyć, że każdy tworzony moduł posiada sygnał gotowości do przyjęcia nowych danych, a danym wyjściowym towarzyszą sygnały, które potwierdzają ich ważność. Tak więc każdy kolejny komponent w łańcuchu przetwarzania zaczyna pracować, gdy wszystkie dane wejściowe posiadają odpowiednie sygnały gotowości, a sam komponent jest dostępny dla przetwarzania następnej porcji danych wejściowych. Mechanizm ten umożliwia implementację algorytmów, które zakładają zmienną liczbę taktów potrzebną do zakończenia pracy, więc ostateczne opóźnienie całego łańcucha przetwarzania jest również zmienne.

7.8 Ocena wydajności

Cała implementacja została przeprowadzona na SoC (ang. *System-on-Chip*) Zynq firmy Xilinx. W związku z tym porównanie dotyczy wydajności osiągniętej w systemie osadzonym z ARM Cortex-A9 (zarówno z przyspieszeniem FPGA jak i bez) oraz na typowej stacji roboczej z procesorem Intel Core i7. Systemy oparte na procesorach (Intel Core i7-4770 i ARM Cortex-A9) wykorzystują pojedynczą precyzję (32 bity), podczas gdy przyspieszenie oparte na FPGA wykorzystuje połówkową reprezentację (16 bitów) liczb zmiennoprzecinkowych. Pomimo różnic w reprezentacji liczbowej, stosowanie się do ograniczenia (6.14) wynikającego z lematu o długości sekwencji bezpiecznej numerycznie gwarantuje stabilność obliczeń. Dla obu procesorów przeprowadzono obliczenia w oparciu o kod skompilowany przez gcc 4.6 dla pojedynczego rdzenia. Na obu procesorach został przetestowany ten sam program (bez specjalnej optymalizacji). Czas wykonania algorytmów związanych z modelem HMM mierzono na podstawie obliczenie syntetycznej struktury HMM z 30 stanami i 1000 obserwacji oraz sekwencją obserwacji o długości $T = 200$. Jako wskaźnik efektywności przyjęto czas potrzebny do pełnego zbadania danej sekwencji obserwacji za pomocą określonego modelu. To podejście daje przybliżoną ocenę w perspektywie rzeczywistych aplikacji. Porównanie wydajności obliczeniowej dla zlogarytmowanych postaci algorytmów Bauma-Welcha D&C i Viterbiego D&C przedstawia odpowiednio Tabela 7.7 oraz Tabela 7.8.

Etapy Baum-Welch	Intel Core i7-4770	ARM Cortex-A9	Logika FPGA D&C
Forward-Backward	31.96	345.67	3.31
Gamma-Xi	17.75	195.04	2.14
Numerator-Denominator	16.48	176.94	1.59
Cały Baum-Welch	66.19	717.65	7.03 (4.89)

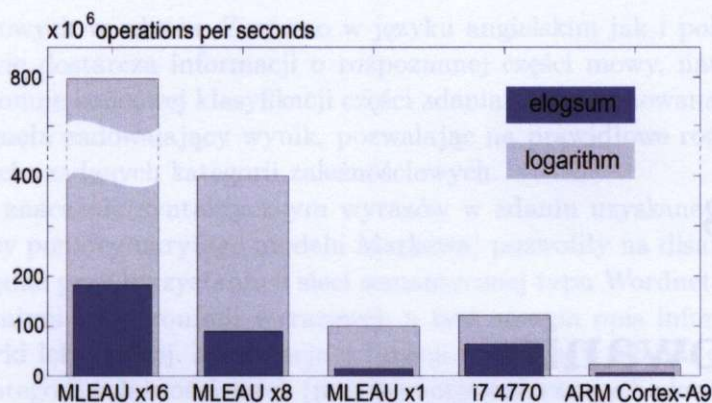
Tabela 7.7: Porównanie wydajności obliczania algorytmu Baum-Welch D&C, czas w [ms].
(Źródło: *opracowanie własne*).

Etapy Viterbi	Intel Core i7-4770	ARM Cortex-A9	FPGA D&C akceleracja	FPGA pełna równoległość
Inicjalizacja	1.3 236	9.3 750	0.7 875	0.0 158
Indukcja	3.2 766	15.2 877	0.9 510	0.0 210
Nawracanie	1.76 e -4	7.50 e -4	3.20 e -4	3.20 e -4
Cały Viterbi	4.6 004	24.6 635	1.7 388	0.0 371

Tabela 7.8: Porównanie wydajności obliczania algorytmu Viterbiego D&C, czas w [ms].
(Źródło: *opracowanie własne*).

7.8.1 MLEAU

W obliczeniach związanych z modelami HMM wykonywanych w dziedzinie logarytmów kluczowa arytmetyka obejmuje sekwencję operacji rozszerzonych: porównanie logarytmiczne, wstępną sumę logarytmów, konwersję na wykładnik, konwersję na logarytm, końcową sumę logarytmów. W związku z tym, w celu oceny wydajności, mierzony jest czas wykonania konwersji logarytmicznej, jak i czas wykonania rozszerzonej sumy logarytmów (elogS) na różnych platformach. Konfiguracja porównywanych procesorów jest taka sama jak przy porównaniu wydajności algorytmu Bauma-Welcha D&C i Viterbiego D&C. Moduł MLEAU zrealizowany w FPGA taktowany jest zegarem 200 MHz, a dla pokazania wydajności przy różnym stopniu zrównoleglenia liczba interfejsów wynosi 1, 8 i 16 (patrz Rysunek 7.21).



Rysunek 7.21: Porównanie wydajności rozszerzonej arytmetyki w dziedzinie logarytmów.
(Źródło: opracowanie własne).

7.8.2 Wpływ zredukowania precyzji zmiennoprzecinkowej modeli na długość badanej sekwencji

Dla wszystkich przedstawionych ukrytych modeli Markowa określona została, względem zredukowanych precyzji zmiennoprzecinkowych, maksymalna długość badanej sekwencji do której zagwarantowane jest bezpieczeństwo numeryczne obliczeń. Zestawienie wpływu zastosowania lematu dla modeli HMM dedykowanych dla syntaktycznej i semantycznej analizy danych tekstowych (wyliczanych w połówkowej precyzji obliczeniowej) przedstawia Tabela 7.9.

Nazwa ukrytego modelu Markowa	Bezpieczna długość badanej sekwencji	Procent błędów rozpoznawania bez uwzględnienia lematu	Procent błędów rozpoznawania z uwzględnieniem lematu
Model z obserwacją agregowaną binarnie dla ekstrakcji z HTML	1 048	5.35% (824/15 400)	0.26% (41/15 400)
Model z obserwacją agregowaną wzrażeńiowo dla ekstrakcji z HTML	121	7.79% (1 201/15 400)	0.49% (76/15 400)
Model rozpoznawania części mowy dla j. angielskiego	213	18.70% (8 859/47 370)	0.26% (124/47 370)
Model rozpoznawania grup frazowych dla j. angielskiego	925	21.50% (10 194/47 400)	0.25% (120/47 400)
Model rozpoznawania części zdania dla j. angielskiego	32	77.40% (44 648/57 680)	0.44% (256/57 680)
Model rozpoznawania części mowy dla j. polskiego	332	16.71% (12 396/74 182)	0.39% (295/74 182)
Model rozpoznawania części zdania dla j. polskiego	37	21.35% (2 285/10 702)	0.64% (69/10 702)

Tabela 7.9: Zestawienie wpływu zastosowania lematu dla modeli HMM prezentowanych w rozprawie wyliczanych w połówkowej precyzji obliczeniowej.
(Źródło: opracowanie własne).

Nie uwzględniając lematu, rozpoznawanie zostało wykonane na pełnej długości sekwencji obserwacji (np. o długości 15 400). Natomiast przy uwzględnieniu lematu długość badanej sekwencji obserwacji była nie dłuższa niż bezpieczna długość sekwencji wskazane przez zastosowanie lematu dla danego modelu. Pomimo zastosowania lematu nadal w obliczeniach o zredukowanej precyzji może dochodzić do błędów zaokrągleń, niemniej jednak wyniki przedstawione w Tabeli 7.9 dowodzą, że zastosowanie lematu jest konieczne dla zmniejszenia liczby błędnie rozpoznanych kategorii.

Rozdział 8

Podsumowanie

W niniejszej rozprawie przedstawiona została sprzętowa implementacja algorytmów programowania dynamicznego stosowanych w ukrytych modelach Markowa przeznaczonych do syntaktycznej i semantycznej analizy danych tekstowych. Na potrzeby prac badawczych opracowana została aplikacja HMM-Toolbox, pozwalająca na tworzenie, zarządzanie, uczenie i testowanie ukrytych modeli Markowa.

Dla pozyskiwania istotnych danych tekstowych ze źródeł HTML zaprojektowano model Markowa, proponując dwa style obserwacji z przestrzeni międzytekstowej: znaczniki grupowane binarnie i znaczniki grupowane w wyrażenia. Nauczanie modeli przeprowadzono etapami w sposób nadzorowany z uwzględnieniem kroku związanego z treningiem Viterbiego. Etap znakowania treści witryn internetowych przeprowadzono z wykorzystaniem funkcji zbiorczego oznaczania treści w aplikacji HMM-Toolbox. Oba modele wykazały bardzo wysoką dokładność rozpoznawania treści właściwej (powyżej 98%), przy czym obserwacja oparta o grupowanie binarne lepiej radziła sobie przy klasyfikacji nieznanymi przestrzeni międzytekstowych. Cecha ta jest o tyle ważna, że przy ocenie zmieniającej się w czasie struktury witryny internetowej nauczony model przez dłuższy okres dostarcza poprawnych wyników klasyfikacji. Opracowana metoda pozyskiwania istotnych danych ze źródeł HTML bazuje głównie na przestrzeni międzytekstowej (zestawy znaczników HTML), a tym samym jest niezależna od języka witryny internetowej.

W zadaniach oznaczania tekstów częściami mowy i częściami zdania opracowano wiele autorskich modeli Markowa różniących się zarówno przestrzenią stanów jak i zestawem możliwych obserwacji. Niemniej jednak dla wszystkich modeli zaproponowano jednolity styl obserwacji na podstawie morfemów brzegowych tworzonych przez zadanej długości afiksy wyrazów z możliwością stopniowego wygaszania wcześniejszych afiksów. Ten rodzaj obserwacji pozwala na określenie precyzji odwzorowania wyrazów, co pozwala na zredukowanie liczności dla n -gramowego zestawu obserwacji.

Przy oznaczaniu tekstu częściami mowy obserwacje w ukrytym modelu Markowa stanowiły morfemy brzegowe będące nośnikiem informacji o znaczeniu syntaktycznym wyrazu. Nauczanie modeli wykonane zostało w oparciu o bank drzew składniowych (dla języka angielskiego Penn Treebank, a dla języka polskiego Składnica frazowa). Dla obydwu języków osiągnięto wysoką dokładność rzędu 95% dla wyrazów grupowanych w ramach znanych afiksów.

Dla języka angielskiego dokonany został podział zdania na części stanowiące grupy frazowe. Dokonano oceny skuteczności rozpoznawania grup frazowych przez pojedynczy jak i dwuwarstwowy ukryty model Markowa. W pojedynczym modelu przestrzeń stanów i obserwacji bezpośrednio wynika z bazy danych uczących. Dla dwuwarstwowego ukrytego modelu Markowa, przestrzeń obserwacji na wyższym poziomie zawiera informacje zarówno z afiksów wyrazów, jak i informacje o części mowy odkrytej przez model Markowa z warstwy niższej. Dokładność klasyfikacji czterech najbardziej znaczących grup frazowych z wykorzystaniem dwuwarstwowego modelu wyniosła około 93%, podczas gdy stosując pojedynczy model, otrzymywany wynik rozpoznawania był o 10 punktów procentowych słabszy.

Warstwowe ukryte modele Markowa zostały zastosowane również przy rozpoznawaniu typów

relacji zależnościowych w zdaniu. Zarówno w języku angielskim jak i polskim model Markowa w niższej warstwie dostarcza informacji o rozpoznanej części mowy, natomiast model z wyższej warstwy dokonuje końcowej klasyfikacji części zdania. Zaproponowana warstwowa struktura klasyfikacji osiągnęła zadowalający wynik, pozwalając na prawidłowe rozpoznanie co najmniej siedmiu z dziesięciu zadanych kategorii zależnościowych.

Informacje o znaczeniu syntaktycznym wyrazów w zdaniu uzyskane w wyniku klasyfikacji części mowy (przy pomocy ukrytego modelu Markowa) pozwoliły na disambiguację syntaktyczną, co jest niezbędne przy korzystaniu z sieci semantycznej typu Wordnet. Umożliwiło to trafne odkrywanie hiponimii i hiperonimii wyrazowej, a tym samym opis informacji na różnych poziomach semantyki leksykalnej. Informacje o funkcji wyrazów w zdaniu dostarczone w wyniku przypisania im kategorii zależnościowych (przy pomocy warstwowych ukrytych modeli Markowa) pozwoliły na odkrywanie relacji semantycznych niezbędnych przy tworzeniu opisu semantycznego.

Przedstawione algorytmy programowania dynamicznego wykorzystywane w obliczeniach ukrytych modeli Markowa zostały z powodzeniem zrealizowane sprzętowo w logice programowalnej układu FPGA. Struktura algorytmiczna została utworzona wykorzystując syntezę na wysokim poziomie HLS, natomiast kosztowne obliczeniowo przekształcenia logarytmiczne i wykładnicze realizowane są według autorskiego pomysłu przez wyspecjalizowaną autorską jednostkę aproksymacji MLEAU utworzoną w języku opisu sprzętu. Tym samym powstała architektura zapewnia wysoką wydajność obliczeniową przy jednoczesnym zachowaniu elastyczności. Dzięki HLS możliwe jest łatwe dopasowanie architektury do ilości i rodzaju alokowanej pamięci, wykorzystywanej precyzji obliczeniowej oraz stopnia zrównoleglenia. Uzyskane w ten sposób stopnie swobody pozwalają już we wstępnej fazie projektowania na odpowiednie względem potrzeb aplikacji balansowanie pomiędzy:

1. czasem wykonania obliczeń,
2. zapotrzebowaniem na zasoby sprzętowe FPGA,
3. poborem mocy układu.

Parametry te są ze sobą sprzężone i zmiana jednego wpływa oczywiście na pozostałe. Jeżeli celem jest krótki czas wykonania obliczeń, to jest to dokonane zwiększonym zapotrzebowaniem na zasoby jak i zwiększonym poborem mocy układu. Jeżeli natomiast celem jest oszczędne zagospodarowanie zasobów logiki FPGA, to obliczenia wykonywane będą bardziej sekwencyjnie i w dłuższym czasie. W celu efektywnego wykonania przekształceń logarytmicznych i wykładniczych koniecznych przy wyliczaniu stabilnych numerycznie ukrytych modeli Markowa zaproponowano technikę dostępu do tablicy przeglądowej wykorzystującą blisko asocjacyjne mapowania wejść i wyjść. Zabieg ten pozwala na implementację pojedynczej złożonej tablicy przeglądowej, której wartości dostępne są prawie równolegle dla wszystkich żądań. Konflikty żądań rozwiązywane są przez tabele kodów stopowych. W wyniku powstałych opóźnień dostęp do tablicy przy 16 równoczesnych żądaniach mógł wynieść 1 cykl, gdy wszystkie żądania zasobów nie powodowały konfliktu (pełna równoległość) i aż do 16 cykli, gdy wszystkie żądania wskazywały na tę samą pulę wartości, powodując konflikt globalny. Podczas badania nad wykorzystaniem tej techniki w logarytmicznej wersji algorytmu Forward-Backward wystąpienia opóźnień przekształceń logarytmicznych nie przekraczały 33% niezależnie od stopnia zrównoleglenia, a średni czas dostępu przy 16 równoczesnych żądaniach wynosił 3 cykle zegarowe.

Przeprowadzanie efektywnych i bezpiecznych numerycznie obliczeń algorytmów dedykowanych dla ukrytych modeli Markowa o zredukowanej reprezentacji zmiennoprzecinkowej wymaga nałożenia pewnych ograniczeń na długość badanej sekwencji obserwacji. Przedstawiony i udowodniony w rozprawie lemat o bezpieczeństwie numerycznym ukrytych modeli Markowa nadaje ramy obliczeniowe gwarantujące stabilność numeryczną i jednocześnie stanowi uzasadnienie stawianej w rozprawie tezy.

Spis rysunków

1.1	Składniowa struktura zdania z uwzględnieniem morfologii wyrazów	8
2.1	Ukryty model Markowa z n -gramową strukturą obserwacji	13
2.2	Warstwowa struktura przetwarzania ukrytych modeli Markowa	14
2.3	Przykładowa zawartość zakładki „Creation“ do tworzenia nowych modeli HMM w aplikacji HMM-Toolbox	23
2.4	Nauczanie i weryfikacja modelu w HMM-Toolbox	25
2.5	Przykład zdania w HMM-Toolbox z przypisanymi kategoriami syntaktycznymi do poszczególnych wyrazów.	25
2.6	Zakładka wizualizacji struktury drzewa dla pojedynczego zdania w HMM-Toolbox	26
3.1	Przykładowa witryna internetowa z oznaczoną treścią właściwą artykułu oraz treścią dodatkową	28
3.2	Przykładowe witryny internetowe z oznaczoną treścią właściwą artykułów oraz treścią dodatkową	29
3.3	Przykład hierarchii w obiektowym modelu dokumentu HTML	33
3.4	Wstępna sekwencja głównych kategorii w formacie HTML	35
3.5	Grupowanie informacji z przestrzeni międzytekstowej	36
3.6	Funkcja zbiorczego oznaczania treści w HMM-Toolbox	38
3.7	Procedura pozyskiwania danych z witryn internetowych oraz nauczanie i weryfikacja modelu HMM	39
3.8	Tryb podglądu pozyskiwanej informacji z witryny internetowej w HMM-Toolbox	40
3.9	Wizualizacja rozkładu prawdopodobieństwa dla obserwacji grupowanej binarnie	42
3.10	Wizualizacja rozkładu prawdopodobieństwa dla obserwacji grupowanej w wyrażenia	43
3.11	Model Markowa dla klasyfikacji artykułów z portalu www.onet.pl	43
4.1	Przykład drzewa składniowego dla języka angielskiego	45
4.2	Przykład drzewa składniowego dla języka polskiego	46
4.3	Tabela odkrywania wyrażen nieterminalnych algorytmem CYK	48
4.4	Przykład rozpoznawania kategorii syntaktycznych w zdaniu z wykorzystaniem ukrytego modelu Markowa	60
4.5	Model Markowa prezentujący prawdopodobieństwa przejść między wybranymi kategoriami syntaktycznymi	60
4.6	Przykład rozpoznawania grup frazowych w wykorzystaniem ukrytego modelu Markowa	63
4.7	Model Markowa prezentujący prawdopodobieństwa przejść między wybranymi grupami frazowymi	64
4.8	Przykład rozpoznawania funkcji wyrazów w zdaniu z wykorzystaniem ukrytego modelu Markowa	67
4.9	Model Markowa prezentujący prawdopodobieństwa przejść między wybranymi kategoriami funkcji wyrazów w zdaniu	67
4.10	Przykład rozpoznawania kategorii syntaktycznych z wykorzystaniem ukrytego modelu Markowa	72

4.11	Model Markowa prezentujący prawdopodobieństwa przejść między wybranymi kategoriami syntaktycznymi	73
4.12	Przykład rozpoznawania kategorii zależnościowej z wykorzystaniem ukrytych modeli Markowa	76
4.13	Model Markowa prezentujący prawdopodobieństwa przejść między wybranymi kategoriami zależnościami	76
4.14	Przykład rozpoznawania kategorii zależnościami z wykorzystaniem warstwowych ukrytych modeli Markowa	77
4.15	Model Markowa prezentujący prawdopodobieństwa przejść między wybranymi kategoriami zależnościami	77
4.16	Zakładka wpisywania i rozpoznawania nowego tekstu w HMM-Toolbox	79
5.1	Graficzna reprezentacja relacji między wyrazami w zdaniu	83
5.2	Graficzna reprezentacja wartości dopasowania ciągów literowych w zależności od długości ciągu	86
5.3	Opis informacji na różnych poziomach semantyki leksykalnej w HMM-Toolbox	87
6.1	Szczególny model HMM dla lematu	90
6.2	Widok zakładki do redukcji reprezentacji liczbowej w ramach aplikacji HMM-Toolbox	95
7.1	Uproszczona ścieżka projektowania w Vivado HLS	99
7.2	Pojęcia opóźnienia i przepustowości w FPGA	100
7.3	Podstawy Vivado HLS – sterowanie syntezą za pomocą dyrektyw	101
7.4	Zasady automatycznego tworzenia sterujących automatów w Vivado HLS	102
7.5	Zynq – programowalny układ scalony będący połączeniem procesora i FPGA	106
7.6	Dostęp do pamięci jedno- i wielo-portowej	108
7.7	Modyfikacje dostępu do macierzy wielowymiarowej mapowanej na pamięci wieloportowej	108
7.8	Przypisanie wykonywanych zadań do zasobów sprzętowych	110
7.9	Ogólna koncepcja przetwarzania HMM z wykorzystaniem SPU	111
7.10	Wizualizacja bloku algorytmicznego przeznaczonego pod obliczenia związane z algorytmem Forward-Backward D&C	112
7.11	Wizualizacja bloku algorytmicznego przeznaczonego pod obliczenia związane z algorytmem Gamma-Xi D&C	113
7.12	Wizualizacja bloku algorytmicznego przeznaczonego pod obliczenia związane z algorytmem Numerator-Denominator D&C	115
7.13	Wizualizacja bloku algorytmicznego przeznaczonego pod obliczenia związane z algorytmem Viterbi D&C	117
7.14	Nawracanie w przypadku pełnej zgodności kawałków, jak i w przypadku niedopasowania granic kawałków	119
7.15	Ogólna architektura bloku aproksymacji dla przekształcenia funkcji logarytmicznej i wykładniczej	121
7.16	Wizualizacja zawartości tablic przeglądowych miniLUT dedykowanych dla mantysy w przekształceniu logarytmicznym	122
7.17	Przykład operacji aproksymacji przekształcenia dla funkcji logarytmicznej	122
7.18	Wizualizacja zawartości tablic przeglądowych miniLUT dedykowanych dla mantysy w przekształceniu wykładniczym	124
7.19	Przykład operacji aproksymacji przekształcenia dla funkcji wykładniczej	124
7.20	Blisko asocjacyjne mapowanie wejścia i wyjścia dla zrównoleglenia konwersji LUT	126
7.21	Porównanie wydajności rozszerzonej arytmetyki w dziedzinie logarytmów	131

Spis tabel

3.1	Zestaw mierzalnych cech w przestrzeni tekstowej	37
3.2	Zestaw kategorii przypisanych do stanów modelu	37
3.3	Precyzja klasyfikacji informacji z artykułów portalu www.onet.pl z wykorzystaniem obserwacji grupowanej w wyrażenia	41
3.4	Precyzja klasyfikacji informacji z artykułów portalu www.onet.pl z wykorzystaniem obserwacji grupowanej binarnie	42
4.1	Zastosowanie obserwacji afiksowej w wariancie Uni-, Bi- i Tri-gramowym	56
4.2	Pokrycie słownika Wordnet w odniesieniu do różnych długości znaków w afiksach	56
4.3	Kategorie syntaktyczne dla języka angielskiego przypisane do stanów w ukrytym modelu Markowa	58
4.4	Precyzja rozpoznawania kategorii syntaktycznych z wykorzystaniem ukrytego modelu Markowa dedykowanego dla języka angielskiego	59
4.5	Kategorie grup frazowych dla języka angielskiego przypisane do stanów w ukrytym modelu Markowa	61
4.6	Dokładność rozpoznawania grup frazowych z wykorzystaniem ukrytego modelu Markowa dedykowanego dla języka angielskiego	62
4.7	Dokładność rozpoznawania grup frazowych z wykorzystaniem warstwowego ukrytego modelu Markowa dedykowanego dla języka angielskiego	63
4.8	Kategorie funkcji wyrazów w zdaniu dla języka angielskiego przypisane do stanów w ukrytym modelu Markowa	65
4.9	Dokładność rozpoznawania funkcji wyrazów w zdaniu z wykorzystaniem ukrytego modelu Markowa dedykowanego dla języka angielskiego	66
4.10	Kategorie syntaktyczne dla języka polskiego przypisane do stanów w ukrytym modelu Markowa	69
4.11	Dokładność rozpoznawania kategorii syntaktycznych z wykorzystaniem ukrytego modelu Markowa dedykowanego dla języka polskiego	70
4.12	Kategorie syntaktyczne rozszerzone o rozpoznawanie przypadków, osoby, przymiotnikowości i aspektu dla języka polskiego przypisane do stanów w modelu Markowa	71
4.13	Kategorie syntaktyczne rozszerzone o rozpoznawanie liczby, rodzaju, stopnia i akomodacyjności dla języka polskiego przypisane do stanów w modelu Markowa	71
4.14	Kategorie syntaktyczne rozszerzone o rozpoznawanie negacji, akcentowości, aglutynacyjności i wokaliczności dla języka polskiego przypisane do stanów w modelu Markowa	72
4.15	Kategorie zależnościowe dla języka polskiego przypisane do stanów w ukrytym modelu Markowa	74
4.16	Dokładność rozpoznawania kategorii zależnościowej z wykorzystaniem ukrytego modelu Markowa dedykowanego dla języka polskiego	75
4.17	Porównanie miar F1 dla systemów rozpoznawania kategorii syntaktycznych dla języka angielskiego	78
4.18	Porównanie precyzji dla systemów rozpoznawania typu zależności syntaktycznych i semantycznych dla języka angielskiego	78

4.19	Porównanie precyzji dla systemów rozpoznawania kategorii syntaktycznych dla języka polskiego	78
4.20	Porównanie precyzji dla systemów rozpoznawania kategorii zależnościowych dla języka polskiego	79
5.1	Zestawienie funkcji pełnionych przez związki wyrazowe	81
5.4	Wartość dopasowania ciągów literowych, gdzie zmianie ulega kombinacja znaków na różnej pozycji	86
6.1	Długości sekwencji stabilne numerycznie dla różnej precyzji reprezentacji liczbowej	96
7.1	Wyniki syntezy projektu	104
7.2	Przykład żądań uzyskania dostępu do wybranych zasobów	126
7.3	Mapa żądań dostępu do zasobów względem wartości wejściowej	127
7.4	Mapa konfliktów dostępu do zasobów względem wartości wejściowej	127
7.5	Kody stopowe określające blokowane wejścia w danej iteracji	128
7.6	Wykorzystanie zasobów FPGA w zależności od stopnia zrównoleglenia	129
7.7	Porównanie wydajności obliczania algorytmu Baum-Welch D&C	130
7.8	Porównanie wydajności obliczania algorytmu Viterbiego D&C	130
7.9	Zestawienie wpływu zastosowania lematu dla modeli HMM prezentowanych w rozprawie wyliczanych w połówkowej precyzji obliczeniowej	131

Bibliografia

- Abney, S. & Light, M. (1999), Hiding a Semantic Class Hierarchy in a Markov Model, in 'Proceedings of ACL Workshop on Unsupervised Learning in NLP. University of Maryland'.
- Acedański, S. (2010), 'A morphosyntactic brill tagger for inflectional languages', H. Loftsson, E. Roegnvaldsson, S. Helgadóttir (eds.) *Advances in Natural Language Processing LNCS* 6233, 3–14.
- Aithal, S. (2016), 'A survey on partial reconfiguration techniques', *International Journal Of Engineering And Computer Science* 5(5), 16613–16616.
- Andersen, P. M., Hayes, P. J., Huettnner, A. K., Schmandt, L. M., Nirenburg, I. B. & Weinstein, S. P. (1992), Automatic extraction of facts from press releases to generate news stories, in 'Proceeding ANLC '92 Proceedings of the third conference on Applied natural language processing', Trento, Italy, pp. 170–177.
- Anisha, A. T. & C.Sunitha (2015), 'A hybrid parts of speech tagger for malayalam language', *Advances in Computing, Communications and Informatics (ICACCI)*, 2015 International Conference pp. 1502–1507.
- Bapat, O. A., Fastow, R. M. & Olson, J. (2013), 'Acoustic coprocessor for hmm based embedded speech recognition systems', *IEEE Transactions on Consumer Electronics* 59, 629–633.
- Baroni, M. & Zamparelli, R. (2010), Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space, in 'Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing', Cambridge, Massachusetts, pp. 1183–1193.
- Behnam, R. E. F. (2015), 'Stats-calculus pose descriptor feeding a discrete HMM low-latency detection and recognition system for 3D skeletal actions'.
- Bjorkelund, A., Bohnet, B., Hafdell, L. & Nugues, P. (2010), A high-performance syntactic and semantic dependency parser, in 'Proceedings of the 23rd International Conference on Computational Linguistics: Demonstrations', Beijing, China, pp. 33–36.
- Bąk, P. (2004), *Gramatyka języka polskiego: zarys popularny*, Wiedza Powszechna.
- Bos, J. (2011), 'A survey of computational semantics: Representation, inference and knowledge in wide-coverage text understanding', *Language and Linguistics Compass* 5/6, 336–366.
- Brants, T. (2000), Tnt: A statistical part of speech tagger, in 'Proceeding ANLC '00 Proceedings of the sixth conference on Applied natural language processing', Seattle, Washington, USA, pp. 224–231.
- Brown, D. & Golod, D. (2009), 'A tutorial of techniques for improving standard hidden Markov model algorithms', *J. Bioinformatics and Computational Biology* pp. 737–754.
- Cambria, E. & White, B. (2014), 'Jumping nlp curves: A review of natural language processing research', *IEEE Computational Intelligence Magazine* 9, 48–57.

- Carreras, X. (2007), Experiments with a higher-order projective dependency parser, in 'Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007', Prague, Czech Republic, pp. 957–961.
- Charwat, G., Dvorak, W., Gaggl, S. A., Wallner, J. P. & Woltran, S. (2015), 'Methods for solving reasoning problems in abstract argumentation - A survey', *Journal Artificial Intelligence* 220, 28–63.
- Choi, J. D. & Palmer, M. (2011), Transition-based semantic role labeling using predicate argument clustering, in 'Proceedings of the ACL 2011 Workshop on Relational Models of Semantics', Portland, Oregon, pp. 37–45.
- Choi, Y., You, K., Choi, J. & Sung, W. (2010), 'A real-time FPGA-based 20,000-word speech recognizer with optimized DRAM access', *IEEE Transactions on Circuits and Systems Part I: Regular Papers* 27, 2119–2131.
- Chomsky, N. (1956), 'Three models for the description of language', *IRE Transactions on Information Theory* 2, 113–124.
- Chomsky, N. (1957), *Syntactic Structures*, Mouton, The Hague, Paris.
- Chrysos, G. & i in. (2012), 'Opportunities from the use of FPGAs as platforms for bioinformatics algorithms', *International Conference on Bioinformatics and Bioengineering (BIBE)*, 2012 IEEE 12th pp. 559–565.
- Churbanov, A. & Winters-Hilt, S. (2008), 'Implementing EM and Viterbi algorithms for hidden Markov model in linear memory', *BMC Bioinformatics* 9, 1–15.
- Ciecierski, T. (2011), *Zależność kontekstowa: wprowadzenie do problematyki*, Ośrodek Badań Filozoficznych.
- Cohn, T. & Blunsom, P. (2005), Semantic role labelling with tree conditional random fields, in 'Proceedings of the Ninth Conference on Computational Natural Language Learning', Ann Arbor, Michigan, pp. 169–172.
- Collins, M. (2002), Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms, in 'Proceedings of the ACL-02 conference on Empirical methods in natural language processing', pp. 1–8.
- Collins, M. (2003), 'Head-driven statistical models for natural language parsing', *Computational linguistics* 29, 589–637.
- Courcelle, B. (2006), 'The monadic second-order logic of graphs xvi : Canonical graph decompositions', *Logical Methods in Computer Science* 2, 1–46.
- Covington, M. A. (2001), A fundamental algorithm for dependency parsing, in 'Proceedings of the 39th annual ACM Southeast Conference', pp. 95–102.
- Crescenzi, V. & Mecca, G. (2004), 'Automatic information extraction from large websites', *Journal of the ACM* 51, 731–779.
- Dai, Q., Chen, E. & Shi, L. (2009), An iterative approach for joint dependency parsing and semantic role labeling, in 'Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task', Boulder, Colorado, pp. 19–24.
- Dhawan, U. & DeHon, A. (2013), Area-efficient near-associative memories on FPGAs, in 'Proceeding FPGA '13 Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays', Monterey, California, USA, pp. 191–200.

- Dhawan, U. & Dehon, A. (2015), 'Area-Efficient Near-Associative Memories on FPGAs', *ACM Transactions on Reconfigurable Technology and Systems (TRETs)* 7, 30–41.
- Dipasquo, D. (1998), 'Using HTML Formatting to Aid in Natural Language Processing on the World Wide Web', School of Computer Science. Senior Honors Thesis.
- Earley, J. (1970), 'An efficient context-free parsing algorithm', *Communications of the ACM* 13, 94–102.
- Eisner, J. M. (1996), Three new probabilistic models for dependency parsing: An exploration, in 'Proceedings of the 16th conference on Computational linguistics', Copenhagen, Denmark, pp. 340–345.
- Ellson, J., Gansner, E., Koutsofios, L., North, S., Woodhull, G., Description, S. & Technologies, L. (2001), Graphviz — open source graph drawing tools, in 'Lecture Notes in Computer Science', Springer-Verlag, pp. 483–484.
- Embley, D. W., Campbell, D. M., Jiang, Y. S., Liddle, S. W., Ng, Y. K., Quass, D. W. & Smith, R. D. (1998), A Conceptual-Modeling Approach to Extracting Data from the Web, in 'Proceedings of the 17th International Conference on Conceptual Modeling (ER'98)', Republic of Singapore, pp. 78–91.
- Embley, D. W., Jiang, Y. & Ng, Y. K. (1999), Boundary discovery in Web Documents, in 'Proceeding SIGMOD '99 Proceedings of the 1999 ACM SIGMOD international conference on Management of data', Philadelphia, Pennsylvania, USA, pp. 467–478.
- Fillmore, C. J. (1968), The case for case, in E. Bach & R. T. Harms, eds, 'Universals in Linguistic Theory', Holt, Rinehart and Winston, London.
- Francois, J.-M. (2006), 'JAHMM — An implementation of HMM in Java'.
- Fung, P. & Chen, B. (2004), BiFrameNet: bilingual frame semantics resource construction by cross-lingual induction, in 'Proceedings of the 20th international conference on Computational Linguistics', pp. 931–935.
- Gaifman, H. (1965), 'Dependency systems and phrase-structure systems', *Information and control* 8, 304–337.
- Gao, C. & Xu, B. (2013), 'The application of semantic field theory to english vocabulary learning', *Theory and Practice in Language Studies* 3(11), 2030–2035.
- Gao, J. & Johnson, M. (2008), A comparison of Bayesian estimators for unsupervised Hidden Markov Model POS taggers, in 'Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing', Honolulu, Hawaii, pp. 344–352.
- Ge, R. & Mooney, R. J. (2005), A statistical semantic parser that integrates syntax and semantics, in 'Proceedings of the Ninth Conference on Computational Natural Language Learning', Ann Arbor, Michigan, pp. 9–16.
- Gesmundo, A., Henderson, J., Merlo, P. & Titov, I. (2009), A latent variable model of synchronous syntactic-semantic parsing for multiple languages, in 'Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task', Boulder, Colorado, pp. 37–42.
- Ghaoui, C. (2005), *Encyclopedia of Human Computer Interaction*, Information Science Reference, Hershey, PA.
- Gildea, D. & Jurafsky, D. (2002), 'Automatic labeling of semantic roles', *Computational Linguistics* 28, 245–288.

- Jimenez, J. & Marquez, L. (2003), Fast and accurate part-of-speech tagging: The svm approach revisited, in 'Proceedings of International Conference Recent Advances in Natural Language Processing', pp. 153–163.
- Gottlob, G. & Koch, C. (2001), 'Logic-based Web Information Extraction', *Journal of the ACM* 33, 87–94.
- Grefenstette, E., Blunsom, P., de Freitas, N. & Hermann, K. M. (2014), 'A deep architecture for semantic parsing', arXiv preprint arXiv:1404.7296 .
- Gupta, S., Agrawal, A., Gopalakrishnan, K. & Narayanan, P. (2015a), Deep Learning with Limited Numerical Precision, in 'Proceeding ICML'15 Proceedings of the 32nd International Conference on International Conference on Machine Learning', Lille, France, pp. 1737–1746.
- Gupta, S., Agrawal, A., Gopalakrishnan, K. & Narayanan, P. (2015b), Deep learning with limited numerical precision, in 'ICML'.
- Hajic, J., Ciaramita, M., Johansson, R. & i in. (2009), The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages, in 'Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task', Boulder, Colorado, pp. 1–18.
- Hajnicz, E. (2014), Lexico-Semantic Annotation of Składnica Treebank by means of PLWN Lexical Units, in 'Proceedings of the 7th International WordNet Conference (GWC 2014)', Tartu, Estonia, pp. 23–31.
- Hays, D. G. (1964), 'Dependency theory: A formalism and some observations', *Language* 40, 511–525.
- Hazem, A. & Morin, E. (2013), A Comparison of Smoothing Techniques for Bilingual Lexicon Extraction from Comparable Corpora, in 'Proceedings of the Sixth Workshop on Building and Using Comparable Corpora', Sofia, Bulgaria, pp. 24–33.
- Henderson, J., Merlo, P., Musillo, G. & Titov, I. (2008), A latent variable model of synchronous parsing for syntactic and semantic dependencies, in 'Proceedings of the Twelfth Conference on Computational Natural Language Learning', Manchester, United Kingdom, pp. 178–182.
- Henderson, J., Merlo, P., Titov, I. & Musillo, G. (2013), 'Multi-lingual joint parsing of syntactic and semantic dependencies with a latent variable model', *Computational Linguistics* 39, 949–998.
- Hovy, D. & Hovy, E. (2012), Exploiting Partial Annotations with EM Training, in 'Held at the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2012)', Montreal, Quebec, Canada, pp. 31–38.
- Hsieh, C. H., Chu, H. P. & Huang, Y. H. (2014), 'An HMM-based eye movement detection system using eeg brain-computer interface', *IEEE International Symposium on Circuits and Systems* pp. 662–665.
- Hulden, M. (2012), Treba: Efficient numerically stable em for pfa, in 'ICGI'.
- Ibrahim, A., Elsimary, H., Aljumah, A. & Gebali, F. (2016), 'Reconfigurable hardware accelerator for profile hidden Markov models', *Arabian Journal for Science and Engineering* 41, 3267–3277.
- Iwakura, T. & Okamoto, S. (2008), 'A fast boosting-based learner for feature-rich tagging and chunking', pp. 17–24.

- Johansson, R. & Nugues, P. (2008), Dependency-based syntactic-semantic analysis with PropBank and NomBank, in 'Proceedings of the Twelfth Conference on Computational Natural Language Learning', Manchester, United Kingdom, pp. 183–187.
- Johansson, R. & Nugues, P. A. (2006), A FrameNet-based semantic role labeler for Swedish, in 'Proceeding COLING-ACL '06 Proceedings of the COLING/ACL on Main conference poster sessions', Sydney, Australia, pp. 436–443.
- Jurafsky, D. & Martin, J. H. (2000), *Speech and Language Processing*, Pearson Education.
- Kaljahi, R. S. Z., Foster, J. & Roturier, J. (2014), Semantic role labelling with minimal resources: Experiments with French, in 'Proceedings of the Third Joint Conference on Lexical and Computational Semantics (*SEM 2014)', Dublin, Ireland, pp. 87–92.
- Karlsson, F. (1990), Constraint grammar as a framework for parsing running text, in 'Proceeding COLING '90 Proceedings of the 13th conference on Computational linguistics', Helsinki, Finland, pp. 168–173.
- Kasami, T. (1966), An efficient recognition and syntax analysis algorithm for context-free languages, Technical Report R-257, University of Illinois, Urbana, Illinois.
- Kobyliński, L. (2014), 'Polita: A multitagger for polish', pp. 2949–2954.
- Kobyliński, L. & Kieraś, W. (2016), 'Part of speech tagging for polish: State of the art and future perspectives'.
- Krasnowska-Kieraś, K. (2017), 'Morphosyntactic disambiguation for polish with bi-lstm neural networks', pp. 2949–2954.
- Kubanek, M., Bobulski, J. & Adrjanowicz, L. (2012), 'Characteristics of the use of coupled hidden Markov models for audio-visual Polish speech recognition', *Bulletin of the Polish Academy of Sciences - Technical Sciences* 60, 307–316.
- Kudo, T. & Matsumoto, Y. (2000), Japanese dependency structure analysis based on support vector machines, in 'Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics', Hong Kong, pp. 18–25.
- Kushmerick, N., Weld, D. S. & Doorenbos, R. (1997), Wrapper induction for information extraction, in 'Proceedings of the International Joint Conference on Artificial Intelligence'.
- Lang, J. & Lapata, M. (2011), Unsupervised semantic role induction via split-merge clustering, in 'Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics', Portland, Oregon, USA, pp. 1117–1126.
- Lang, J. & Lapata, M. (2014), 'Similarity-driven semantic role induction via graph partitioning', *Computational Linguistics* 40, 633–669.
- Lecun, Y., Bengio, Y. & Hinton, G. (2015), 'Deep learning', *Nature* 521, 436–444.
- Lember, J. & Koloydenko, A. (2008), 'The Adjusted Viterbi training for hidden Markov models', *Bernoulli* 14, 180–206.
- Leonard E. Baum, Ted Petrie, G. S. & Weiss, N. (1970), 'A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains', *The Annals of Mathematical Statistics* 41(1), 164–171.
- Levenshtein, A. (1966), 'Binary codes capable of correcting deletions, insertions and reversals', *Soviet Physics Doklady* 10, 707–710.

- Lioma, C. (2008), Part of speech n-grams for information retrieval, PhD thesis, University of Glasgow, Glasgow, Scotland, United Kingdom.
- Liu, D. & Gildea, D. (2010), Semantic role features for machine translation, in 'Proceedings of the 23rd International Conference on Computational Linguistics', Beijing, China, pp. 716–724.
- Lluis, X. (2008), Joint Learning of Syntactic and Semantic Dependencies, PhD thesis, Universitat Politecnica de Catalunya, Barcelona.
- Lluis, X., Bott, S. & Marquez, L. (2009), A second-order joint Eisner model for syntactic and semantic dependency parsing, in 'Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task', Boulder, Colorado, pp. 79–84.
- Lluis, X. & Marquez, L. (2008), A joint model for parsing syntactic and semantic dependencies, in 'Proceedings of the Twelfth Conference on Computational Natural Language Learning', Manchester,, pp. 188–192.
- Llus, X., Carreras, X. & Marquez, L. (2013), 'Joint arc-factored parsing of syntactic and semantic dependencies', *Transactions of the Association for Computational Linguistics* 1, 219–230.
- Lombardo, V. & Lesmo, L. (1996), An Earley-type recognizer for dependency grammar, in 'Proceeding COLING '96 Proceedings of the 16th conference on Computational linguistics', Copenhagen, Denmark, pp. 723–728.
- Luong, M.-T., Nakov, P. & Ken, M.-Y. (2010), A hybrid morpheme-word representation for machine translation of morphologically rich languages, in 'Proceeding EMNLP '10 Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing', Cambridge, Massachusetts, pp. 148–157.
- Lyu, Y., Pan, Q., Zhao, C., Zhu, H., Tang, T. & Zhang, Y. (2015), 'A vision based sense and avoid system for small unmanned helicopter', 2015 International Conference on Unmanned Aircraft Systems (ICUAS) pp. 586–592.
- Madhu, G., Govardhan, A. & Rajinikanth, T. (2011), 'Intelligent Semantic Web Search Engines: A Brief Survey', *International journal of Web and Semantic Technology* 2, 158–172.
- Majumder, T., Pande, P. & Kalyanaraman, A. (2014), 'Hardware accelerators in computational biology: application, potential, and challenges', *IEEE Design and Test* 31, 8–81.
- Manandhar, A., Torrione, P. A. & Collins, L. M. (2015), 'Multiple-instance hidden Markov model for gpr-based landmine detection', *IEEE Transactions on Geoscience and Remote Sensing* 53, 1737–1745.
- Mann, T. P. (2006), 'Numerically stable hidden markov model implementation. An HMM scaling tutorial'.
- Mannini, A., Genovese, V. & Sabatini, A. M. (2014), 'Online decoding of hidden markov models for gait event detection using foot-mounted gyroscopes', *IEEE Journal of Biomedical and Health Informatics* 18, 1122–1130.
- Mari, J. F. & Ber, F. L. (2006), 'Temporal and spatial data mining with second-order hidden markov models', *Soft Computing* 10, 406–414.
- Maziarz, M., Piasecki, M., Rudnicka, E., Szpakowicz, S. & Kędzia, P. (2016), plWordNet 3.0 – a Comprehensive Lexical-Semantic Resource, in 'Proceedings of COLING 2016', Osaka, Japan, pp. 2259–2269.

- McCallum, A., Freitag, D. & Pereira, F. C. N. (2000), Maximum entropy Markov models for information extraction and segmentation, in 'Proceeding ICML '00 Proceedings of the Seventeenth International Conference on Machine Learning', pp. 591–598.
- McDonald, R., Pereira, F., Ribarov, K. & Hajic, J. (2005), Non-projective dependency parsing using spanning tree algorithms, in 'Proceeding HLT '05 Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing', Vancouver, British Columbia, Canada, pp. 523–530.
- McDonald, R. & Satta, G. (2007), On the complexity of non-projective data-driven dependency parsing, in 'Proceedings of the 10th International Conference on Parsing Technologies', Prague, Czech Republic, pp. 121–132.
- Meyers Adam, R. & Macleod, C. (2008), 'Nombank v 1.0 ldc2008t23'.
- Meza-Ruiz, I. & Riedel, S. (2009), Jointly identifying predicates, arguments and senses using Markov logic, in 'Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics', Boulder, Colorado, pp. 155–163.
- Michael Benedikt, L. L. & Neven, F. (2007), 'Logical definability and query languages over ranked and unranked trees', *Journal of the ACM* 8, 1–62.
- Mielczuk, I. (1988), *Dependency Syntax: Theory and Practice*, SUNY Press, New York, Albany.
- Montague, R. (1970), 'Universal grammar', *Theoria* 36, 373–398.
- Nakagawa, T. (2007), Multilingual dependency parsing using global features, in 'Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007', pp. 952–956.
- Nakagawa, T., Kudoh, T. & Matsumoto, Y. (2001), Unknown word guessing and part-of-speech tagging using support vector machines, in 'Proceedings of the Sixth Natural Language Processing Pacific Rim Symposium', pp. 325–331.
- Nane, R., Sima, V.-M. & Pilato, C. (2016), 'A Survey and Evaluation of FPGA High-Level Synthesis Tools', *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 1591–1604.
- Narasimhan, M., Viola, P. A. & Shilman, M. (2006), Online decoding of Markov models under latency constraints, in 'Proceeding ICML '06 Proceedings of the 23rd international conference on Machine learning', Pittsburgh, Pennsylvania, USA, pp. 657–664.
- Nivre, J. (2005a), 'Dependency grammar and dependency parsing', MSI report 5133, 1–32.
- Nivre, J. (2005b), *Inductive Dependency Parsing of Natural Language Text*, PhD thesis, School of Mathematics and Systems Engineering, Vaxjo University.
- Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryigit, G., Kulber, S., Marinov, S. & Marsi, E. (2007), 'MaltParser: A language-independent system for data-driven dependency parsing', *Natural Language Engineering* 13, 95–135.
- Nivre, J. & Nilsson, J. (2005), Pseudo-projective dependency parsing, in 'Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics', Ann Arbor, Michigan, pp. 99–106.
- Ovtcharov, K., Ruwase, O., Kim, J.-Y., Fowers, J., Strauss, K. & Chung, E. (2015), 'Accelerating Deep Convolutional Neural Networks Using Specialized Hardware'.

- Pado, S. & Lapata, M. (2009), 'Cross-lingual annotation projection for semantic roles', *Journal of Artificial Intelligence Research* 36, 307–340.
- Palmer, M., Gildea, D. & P., P. K. (2006), 'The proposition bank: An annotated corpus of semantic roles', *Computational linguistics* 31, 71–106.
- Panuccio, A., Bicego, M. & Murino, V. (2002), A hidden Markov model-based approach to sequential data clustering, in 'Proceedings of the Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition', pp. 734–742.
- Partee, B., Meulen, A. T. & Wall, R. (1993), *Mathematical methods in linguistics*, Kluwer Academic Publisher, Dordrecht, Boston, London.
- Pietras, M. (2013), Knowledge extraction and decision rules generation, based on rough set theory in order to increase the direct marketing campaigns effectiveness, in P. Jałowiecki, E. Jałowiecka, P. Łukasiewicz & A. Orłowski, eds, 'Information Systems in Management XVIII', WULS Press, Warsaw, Poland.
- Pietras, M. (2014a), Hardware conversion of neural networks simulation models for neural processing accelerator implemented as fpga-based soc, in '2014 24th International Conference on Field Programmable Logic and Applications (FPL)', pp. 1–4.
- Pietras, M. (2014b), 'Sentence sentiment classification using fuzzy word matching combined with fuzzy sentiment classifier', *Electrical Review - Special issue* pp. 107–111.
- Pietras, M. (2015), Error analysis in the hardware neural networks applications using reduced floating-point numbers representation, in 'Conference: AIP Conference Proceedings', Vol. 1648, p. 660005.
- Pietras, M. (2017), Hidden markov models with affix based observation in the field of syntactic analysis, in J. Kacprzyk, I. E. Fray, J. Pejaś, A. Piegat, S. ya Kobayashi, J. Kacprzyk, I. E. Fray, J. Pejaś, A. Piegat & S. ya Kobayashi, eds, 'Hard and Soft Computing for Artificial Intelligence, Multimedia and Security', Vol. 534, Springer, Cham, pp. 17–26.
- Pietras, M. & Kłesk, P. (2017), 'Fpga implementation of the logarithmic versions of baum-welch and viterbi algorithms for reduced precision hidden markov models', *Bulletin of the Polish Academy of Sciences: Technical Sciences* 65(6), 935–947.
- Piskorski, J. & Yangarber, R. (2013), *Multi-source, Multilingual Information Extraction and Summarization. Theory and Applications of Natural Language Processing*, Springer, Berlin, Heidelberg, Germany.
- Pradhan, S., Ward, W., Hacioglu, K. & Martin, J. H. (2004), Shallow semantic parsing using support vector machines, in 'HLT-NAACL 2004: Main Proceedings. — Association for Computational Linguistics', Boston, Massachusetts, USA, pp. 233–240.
- Przepiórkowski, A., Bańko, M., Górski, R. L. & Lewandowska-Tomaszczyk, B. (2012), in 'Narodowy Korpus Języka Polskiego', Wydawnictwo Naukowe PWN.
- Punyakanok, V., Roth, D. & t. Yih, W. (2008), 'The importance of syntactic parsing and inference in semantic role labeling', *Computational Linguistics* 34, 257–287.
- Pustejovsky, J. (1995), *The Generative Lexicon*, MIT Press, Cambridge, Massachusetts, USA.
- Rabiner, L. R. (1989), A tutorial on hidden Markov models and selected applications in speech recognition, in 'Proceedings of the IEEE', pp. 257–286.
- Radziszewski, A. (2013), 'A tiered crf tagger for polish', pp. 215–230.

- Ralph Weischedel, Sameer Pradhan, L. R. J. K. M. F. M. E.-B. (2013), 'OntoNotes Release 5.0 LDC2013T19. Web Download', Philadelphia: Linguistic Data Consortium.
- Samuelsson, Y., Tackstrom, O., Velupillai, S. & i in. (2008), Mixing and blending syntactic and semantic dependencies, in 'Proceedings of the Twelfth Conference on Computational Natural Language Learning', Manchester, United Kingdom, pp. 248–252.
- Sang, E. F. T. K., Buchholz, S. & Sang, K. (2000), Introduction to the conll-2000 shared task: Chunking, in 'Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning'.
- Santorini, B., of Pennsylvania. Department of Computer, U. & Science, I. (1989), Part-of-speech Tagging Guidelines for the Penn Treebank Project, MS-CIS, Department of Computer and Information Science, School of Engineering and Applied Science.
- Saramak, B. (2015), Wykorzystanie otwartych źródeł informacji w działalności wywiadowczej: historia, praktyka, perspektywy, Wydział Dziennikarstwa i Nauk Politycznych, Uniwersytet Warszawski, Warsaw, Poland.
- Schmidhuber, J. (2015), 'Deep Learning in Neural Networks: An Overview', *Neural Networks* 61, 85–117.
- Seshadri, N. & Sundberg, C.-E. (1994), 'List Viterbi decoding algorithms with application', *IEEE Transactions on Communications* 42, 313–323.
- Shannon, M., Zen, H. & Byrne, W. (2013), 'Autoregressive models for statistical parametric speech synthesis', *IEEE Transactions on Audio, Speech, and Language Processing* (21, 587–597.
- Shen, D. & Lapata, M. (2007), Using semantic roles to improve question answering, in 'Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)', Prague, Czech Republic, pp. 12–21.
- Shieber, S. M. (1987), 'Evidence against the context-freeness of natural language', *The Formal Complexity of Natural Language* 33, 320–334.
- Singh, U. K., Padmanabhan, V. & Agarwal, A. (2014), 'Dynamic classification of ballistic missiles using neural networks and hidden Markov models', *Applied Soft Computing* 19, 280–289.
- Soni, M. & Dakhole, P. K. (2014), FPGA implementation of content addressable memory based information detection system, in 'Communications and Signal Processing (ICCSP)', Melmaruvathur, India.
- Stolz, A. & Hepp, M. (2013), From RDF to RSS and atom: content syndication with linked data, in 'Proceeding HT '13 Proceedings of the 24th ACM Conference on Hypertext and Social Media', Paris, France, pp. 236–241.
- Sun, W., Li, H. & Sui, Z. (2008), The integration of dependency relation classification and semantic role labeling using bilayer maximum entropy Markov models, in 'Proceedings of the Twelfth Conference on Computational Natural Language Learning', Manchester, United Kingdom, pp. 243–247.
- Sun, Y., Li, P., Gu, G., Wen, Y., Liu, Y. & Liu, D. (2009), Accelerating HMM er on FPGAs using systolic array based architecture, in 'Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS '09)', pp. 1–8.

- Surdeanu, M., Harabagiu, S., Williams, J. & Aarseth, P. (2003), Using predicate-argument structures for information extraction, in 'Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics', Sapporo, Japan, pp. 8–15.
- Surdeanu, M., Johansson, R., Meyers, A. & i in. (2008), The CoNLL-2008 shared task on joint parsing of syntactic and semantic dependencies, in 'Proceeding CoNLL '08 Proceedings of the Twelfth Conference on Computational Natural Language Learning', Manchester, United Kingdom, pp. 159–177.
- Taehwan, K. & Keunsung, B. (2011), 'HMM -based underwater target classification with synthesized active sonar signals', IEICE Trans. Fundamentals E94-A, 2039–2042.
- Tatavarty, U. R. (2011), 'Implementation of numerically stable hidden markov model'. UNLV Theses.
- Taylor, A., Marcus, M. & Santorini, B. (2003), 'The penn treebank: An overview'.
- Tesnière, L. (1969), 'Les Elements de syntaxe structurale', La Syntaxe 1, 36–40.
- Thede, S. M. (1998), Predicting part-of-speech information about unknown words using statistical methods, in 'Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics', Montreal, Quebec, Canada, pp. 1505–1507.
- Thompson, C., Levy, R. & Manning, C. (2003), 'A generative model for semantic role labeling', Machine Learning: ECML 2003 2837, 397–408.
- Titov, I. & Henderson, J. (2007), A latent variable model for generative dependency parsing, in 'Proceedings of the 10th International Conference on Parsing Technologies', Prague, Czech Republic, pp. 144–155.
- Tokuda, K., Nankaku, Y., Toda, T., Zen, H., Yamagishi, J. & Oura, K. (2013), 'Speech Synthesis Based on Hidden Markov Models', Proceedings of the IEEE 101, 1234–1252.
- Tomita, M. (1984), LR parsers for natural languages, in 'Proceedings of the 10th international conference on Computational linguistics', Stanford, California, pp. 354–357.
- Toutanova, K., Haghighi, A. & Manning, C. D. (2005), Joint learning improves semantic role labeling, in 'Proceeding ACL '05 Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics', Ann Arbor, Michigan, pp. 589–596.
- Toutanova, K., Klein, D., Manning, C. D. & Singer, Y. (2003), Feature-rich part-of-speech tagging with a cyclic dependency network, in 'Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics', pp. 173–180.
- Tseng, H., Jurafsky, D. & Manning, C. (2005), Morphological features help pos tagging of unknown words across language varieties, in 'Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing', pp. 32–39.
- Tsuruoka, Y. & Tsujii, J. (2005), Bidirectional inference with the easiest-first strategy for tagging sequence data, in 'Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing', pp. 467–474.
- Vargas, F. L., Fagundes, R. D. R. & Junior, D. B. (2001), 'A FPGA-based viterbi algorithm implementation for speech recognition systems', IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'01) 2, 1217–1220.

- Vinyals, O. & Friedland, G. (2008), 'A Hardware-independent fast logarithm approximation with adjustable accuracy', *Multimedia, ISM 2008, Tenth IEEE International Symposium on* pp. 61–65.
- Waszczuk, J. (2012), 'Harnessing the crf complexity with domain-specific constraints.', pp. 2789–2804.
- Weigel, A. & Fein, F. (1994), Normalizing the weighted edit distance, in 'Proceedings of the 12th IAPR International. Conference on', pp. 399–402.
- Wiliński, A., Bera, A., Pietras, M., Zabłocki, M., Sroka, J., Gościewska, K., Łukasz Brzoso, Jarlaczyński, M. & Błaszyński, P. (2015), *Rozpoznawanie wzorców w szeregach czasowych*, SPWI Szczecin, Szczecin, Poland.
- Woliński, M. (2004), *Komputerowa weryfikacja gramatyki Świdzińskiego*, PhD thesis, Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland.
- Wróblewska, A. (2012), 'Polish dependency bank', *Linguistic Issues in Language Technology* 7.
- Wróblewska, A. (2014), *Polish Dependency Parser Trained on an Automatically Induced Dependency Bank*, PhD thesis, Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland.
- Wróblewska, A. (2018), 'Results of the poleval 2018 competition: Dependency parsing shared task', Ogrodniczuk M. and Kobyliński Ł. (eds.) *Proceedings of the PolEval 2018 Workshop*. Institute of Computer Science, Polish Academy of Sciences 7, 11–24.
- Wróblewska, A. & Przepiórkowski, A. (2014), Projection-based annotation of a polish dependency treebank, in 'Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)', Reykjavik, Iceland, pp. 26–31.
- Wróblewska, A. & Woliński, M. (2012), Preliminary Experiments in Polish Dependency Parsing, in P. Bouvry, M. A. K. nad F. Leprevost, M. Marciniak, A. Mykowiecka & H. Rybiński, eds, 'Security and Intelligent Information Systems', Springer Verlag, Warsaw, Poland.
- Xilinx (2014), *Vivado Design Suite User Guide, High-Level Synthesis*, Vol. UG902, Xilinx.
- Xilinx (2017a), *Vivado HLS Optimization, Methodology Guide*, Vol. UG1270, Xilinx.
- Xilinx (2017b), *Zynq-7000 All Programmable SoC Data Sheet: Overview*, Vol. DS190, Xilinx.
- Yamada, H. & Matsumoto, Y. (2003), Statistical dependency analysis with support vector machines, in 'Proceedings of 8th International Workshop on Parsing Technologies', pp. 195–206.
- Young-Sook, J. L., Lim, J., Hwang, Y., Park, S. & Rim, H. (2004), Semantic role labeling using maximum entropy model, in 'In Proceedings of the CoNLL-2004 Shared Task', pp. 122–125.
- Younger, D. H. (1967), 'Recognition and parsing of context-free languages in time n^3 ', *Information and Control* 10, 189–208.
- Yunfei, G. (2017), *Investigating Read/Write Aggregation to Exploit Power Reduction Opportunities Using Dual Supply Voltages*, Master's thesis, School of Engineering and Applied Science, Washington, USA.
- Zhang, C., Li, P., Sun, G., Guan, Y., Xiao, B. & Cong, J. (2015), Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks, in 'Proceeding FPGA '15 Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays', Monterey, California, USA, pp. 161–170.

Ziółko, B. & Skurzok, D. (2010), Word N-Grams for Polish, in 'The Tenth IASTED International Conference on Artificial Intelligence and Applications', Innsbruck.

Zuo, W., Li, P., Chen, D., Pouchet, L.-N., Zhong, S. & Cong, J. (2013), Improving Polyhedral Code Generation for High-Level Synthesis, in 'Proceeding CODES+ISSS '13 Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis', Montreal, Quebec, Canada.

Materialy powstałe w ramach projektu finansowanego przez Litwiskie Data Consortium Nr LDC0170047, w ramach którego został sfinansowany rozwój i wdrożenie systemu do przetwarzania danych w formacie XML dedykowanego pod aplikację HMM-Toolbox.

A.1. HMM-Toolbox

A.1.1. Projekt HMM-Toolbox

Projekt HMM-Toolbox (LDC00047) (w języku angielskim) to aplikacja HMM-Toolbox utworzona została w języku programowania Java i służy do przetwarzania danych w formacie XML dedykowanego pod aplikację HMM-Toolbox.

A.1.2. Użyte modele Markowa

Model HMM przeznaczony do rozpoznawania części nowej w zdaniu. Model do rozpoznawania części nowej w zdaniu. Model do rozpoznawania części nowej w zdaniu.

Model HMM przeznaczony do rozpoznawania części nowej w zdaniu. Model do rozpoznawania części nowej w zdaniu. Model do rozpoznawania części nowej w zdaniu.

A.3. Sprzętowa realizacja w FPGA

A.3.1. Projekt Vivado

Model HMM przeznaczony do rozpoznawania części nowej w zdaniu. Model do rozpoznawania części nowej w zdaniu. Model do rozpoznawania części nowej w zdaniu.

A.3.2. Projekt ILS

Model do rozpoznawania części nowej w zdaniu. Model do rozpoznawania części nowej w zdaniu. Model do rozpoznawania części nowej w zdaniu.

Model do rozpoznawania części nowej w zdaniu. Model do rozpoznawania części nowej w zdaniu. Model do rozpoznawania części nowej w zdaniu.

A.2. Banki Dźwięków

A.2.1. Penn Treebank

Bank dźwięków składający się z 7 milionów wyrazów oraz 7 milionów wyrazów oraz 7 milionów wyrazów.

Dodatek A

Zawartość płyty DVD

A.1 HMM-Toolbox

A.1.1 Projekt Netbeans

Aplikacja HMM-Toolbox utworzona została w języku programowania Java w środowisku Netbeans 8.0. Informacje statystyczne dotyczące projektu:

A.1.2 Ukryte modele Markowa

Modele zapisane są w formacie XML dedykowanym pod aplikację HMM-Toolbox. Ponadto każdy prezentowany model Markowa posiada ilustracje przejść między stanami zapisaną w uniwersalnym formacie dwuwymiarowej grafiki wektorowej.

- Modele HMM przeznaczone do ekstrakcji informacji z źródeł HTML:
 - Model pozyskiwania treści artykułów z witryny www.onet.pl.
 - Model pozyskiwania treści artykułów z witryny www.theguardian.com.
- Modele HMM przeznaczone do rozpoznawania części mowy w zdaniu:
 - Model do rozpoznawania części mowy dla języka angielskiego.
 - Model do rozpoznawania podstawowych części mowy dla języka polskiego.
 - Model do rozpoznawania rozszerzonych części mowy dla języka polskiego.
- Modele HMM przeznaczone do rozpoznawania typów relacji zależnościowych w zdaniu:
 - Model do rozpoznawania grup frazowych dla języka angielskiego.
 - Model do rozpoznawania typów relacji zależnościowych dla języka angielskiego.
 - Model do rozpoznawania typów relacji zależnościowych dla języka polskiego.

A.1.3 Bazy danych uczących

Do każdego utworzonego modelu Markowa przypisana jest baza danych uczących zapisana w formacie XML dedykowanym pod aplikację HMM-Toolbox.

A.2 Banki Drzew

A.2.1 Penn Treebank

Bank drzew składniowych zawierający ponad 7 milionów wyrazów oznakowanych kategoriami syntaktycznymi, a dla ponad 2 milionów wyrazów uwzględniona jest anotacja semantyczna.

A.2.2 OntoNotes Release 5.0

Finalna wersja projektu OntoNotes (udostępniona przez Linguistic Data Consortium Nr: LDC2013T19) zawierająca około 3 miliony semantycznie oznakowanych wyrazów. W skład bazy danych wchodzi teksty artykułów wiadomości, rozmów telefonicznych i radiowych, jak również teksty starego i nowego testamentu.

A.2.3 2009 CoNLL Shared Task Part 2

Materiały zawarte w niniejszej bazie (udostępnionej przez Linguistic Data Consortium Nr: LDC2012T04) umożliwiają identyfikację zależności składniowych oraz identyfikację argumentów i przypisanie funkcji semantycznych względem predykatu. W skład bazy wchodzi fragmenty następujących korpusów:

- Penn Treebank II (LDC95T7) (w języku angielskim): ponad milion słów kluczowych.
- PropBank (LDC2004T14) (w języku angielskim): semantyczna adnotacja tekstu.
- NomBank (LDC2008T23) (w języku angielskim): struktury argumentów względem rzeczowników.
- Chiński Treebank 6.0 (LDC2007T36) (w języku chińskim): ponad milion wyrazów.
- Chiński Proposition Bank 2.0 (LDC2008T07) (w języku chińskim): anotacja ponad 500 tysięcy wyrazów.

A.2.4 Składnica frazowa i zależnościowa

Baza wyrazów tworząca bank drzew składniowych i zależnościowych dla języka polskiego zawierająca 20000 zdań z oznakowaniem leksykalno - semantycznym. Baza danych zapisana jest w formacie ConLL. Dla każdego wyrazu wyróżnić można oznakowanie części mowy jak i części zdania.

A.3 Sprzętowa realizacja w FPGA

A.3.1 Projekt Vivado

Projekt sprzętowej realizacji algorytmów HMM utworzony w środowisku Vivado Design Suite 2016. Informacje statystyczne dotyczące projektu:

A.3.2 Projekt HLS

Projekty modułów utworzonych przez syntezę wysokiego poziomu:

- Moduł logarytmicznej wersji algorytmu Baum-Welch D&C (ze zintegrowanymi funkcjami obliczeniowymi algorytmów Forward-Backward D&C, Gamma-Xi D&C oraz Numerator-Denominator D&C).
- Moduł logarytmicznej wersji algorytmu Viterbi D&C z nawracaniem kawałkami.

A.3.3 Projekt RTL

Projekt IP-Core dla MLEAU utworzony w języku opisu sprzętu VHDL. Do projektu przypisany jest testbench oraz pliki dla symulacji Matlab.

