AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

KATEDRA INFORMATYKI

Paweł Chrząszcz

Automatyczna ekstrakcja i klasyfikacja semantyczna wielosegmentowych jednostek leksykalnych języka naturalnego

Rozprawa doktorska pod kierunkiem
prof. dr hab. Wiesława Lubaszewskiego

Kraków, 2015

Serdecznie dziękuję mojemu Promotorowi ***prof. dr hab. Wiesławowi Lubaszewskiemu*** *za wszelką pomoc, wskazówki i rady udzielone mi podczas kilkuletnich badań naukowych.*

Spis treści

1. Wstęp ... 9

1.1. Tezy pracy ... 13

1.2. Struktura pracy ... 14

2. Stan badań ... 15

2.1. Named Entity Recognition (NER) ... 15

2.1.1. NER dla języka polskiego ... 18

2.2. Rozpoznawanie i ekstrakcja wyrażeń wielosegmentowych (MWE) ... 20

2.2.1. Struktura MWE w języku polskim ... 22

2.3. Wikipedia jako źródło danych leksykalnych i semantycznych ... 23

2.3.1. Wikipedia a NER ... 26

2.3.2. Wikipedia a ekstrakcja MWE ... 27

3. Narzędzia słownikowe ... 29

3.1. Słownik Fleksyjny Języka Polskiego i biblioteka CLP ... 29

3.2. Ograniczenia słownika SFJP ... 31

3.3. Znaczniki morfosyntaktyczne ... 32

3.4. Słownik CLPM ... 33

3.4.1. Znaczniki słownikowe ... 36

4. Ekstrakcja danych z Wikipedii i Wikisłownika ... 37

4.1. Proces ekstrakcji danych z Wikipedii ... 39

4.2. Proces ekstrakcji danych z Wikisłownika ... 42

4.3. Wyniki ekstrakcji danych ... 43

5. Ekstrakcja etykiet semantycznych dla haseł Wikipedii ... 45

5.1. Dane wejściowe i wyjściowe ... 45

5.2. Ekstrakcja hasła i definicji ... 46

5.3. Podział definicji na fragmenty zdania ... 47

5.4. Wyszukiwanie rzeczownika głównego ... 48

5.5. Operatory – pomocnicze elementy etykiety semantycznej ... 49

5.6. Pozostałe elementy etykiety semantycznej ... 49

5.7. Zapisywanie etykiet ... 51
5.8. Testy i wyniki ... 52
6. Problem ekstrakcji wyrazów wielosegmentowych (WW) ... 55
6.1. Struktura rzeczownika wielosegmentowego ... 56
6.2. Ewaluacja wyników ... 57
6.2.1. Format korpusu testowego ... 58
6.2.2. Przebieg testu ... 59
6.2.3. Znaczenie testu ... 60
7. Metoda DM: *Ekstrakcja WW w oparciu o słownik utworzony z Wikipedii* ... 61
7.1. Wzorce słownikowe ... 61
7.2. Tworzenie wzorców słownikowych z haseł Wikipedii ... 62
7.3. Konstrukcja automatu rozpoznającego wzorce słownikowe ... 64
7.4. Rozpoznawanie wzorców słownikowych w tekście ... 69
7.4.1. Rozpoznawanie wzorców zaczynających się w danym segmencie ... 72
7.4.2. Rozpoznawanie wzorców w tekście ... 73
7.5. Przetwarzanie i zapis wyników ... 75
7.5.1. Format zapisu wyników ... 75
7.5.2. Proces przetwarzania i zapisu wyników ... 77
7.6. Testy metody DM ... 78
8. Metoda pDM: *Ekstrakcja WW z wykorzystaniem wzorców odmiany* ... 81
8.1. Automatyczne wyznaczanie wzorca odmiany na podstawie linków przychodzących ... 81
8.1.1. Wyznaczanie możliwości odmiany pojedynczego linku ... 82
8.1.2. Korekta pierwszej litery hasła ... 85
8.1.3. Tworzenie zbioru wzorców odmiany ... 86
8.1.4. Wyznaczanie optymalnego przecięcia wzorców odmiany ... 88
8.1.5. Proces tworzenia i zapisu wzorców odmiany ... 91
8.2. Użycie treści artykułu do wyznaczenia wzorca odmiany ... 92
8.3. Testy metody pDM ... 92
9. Metoda SM: *Ekstrakcja WW w oparciu o wzorce syntaktyczne* ... 95
9.1. Wzorce syntaktyczne wyrazów ... 96
9.2. Statystyka wystąpień wzorca syntaktycznego ... 98
9.2.1. Tworzenie statystyki wystąpień ... 98
9.3. Wzorce syntaktyczne kontekstu ... 101
9.4. Zapis wzorców syntaktycznych ... 103
9.4.1. Przebieg zapisu wzorców ... 105
9.5. Rozpoznawanie wzorców syntaktycznych – metoda SM ... 106

9.5.1. Przebieg rozpoznawania wzorców syntaktycznych ... 108
9.6. Przetwarzanie i zapis wyników ... 109
9.6.1. Proces zapisu wyników ... 110
9.7. Testy metody SM ... 111
10. Metoda SDM: *Ekstrakcja WW w oparciu o rozszerzony słownik* ... 115
10.1. Tworzenie słownika WW z wyników SM ... 115
10.2. Testy metody SDM ... 117
10.3. Test metody łączonej ... 120
10.3.1. Analiza wyników testu ... 121
10.4. Analiza słownika utworzonego przy pomocy SM ... 123
11. Możliwości dalszego rozwoju ... 127
11.1. Dalszy postęp ekstrakcji WW ... 127
11.2. Klasyfikacja semantyczna wyrazów ... 130
11.2.1. Wzorce syntaktyczne a etykiety semantyczne ... 130
11.2.2. Przyszłe prace nad etykietami semantycznymi ... 132
12. Podsumowanie ... 135
12.1. Efekty pracy ... 135
12.1.1. Narzędzia pomocnicze ... 136
12.1.2. Narzędzia do ekstrakcji wyrazów wielosegmentowych ... 137
12.1.3. Narzędzie ekstrahujące etykiety semantyczne ... 138
12.2. Zakończenie ... 138
13. Dodatek: *Przegląd funkcjonalności projektu* ... 139
13.1. Tworzenie słownika CLPM ... 140
13.2. Ekstrakcja danych z Wikipedii ... 142
13.3. Tworzenie etykiet semantycznych ... 144
13.4. Ekstrakcja WW metodą DM ... 145
13.5. Ekstrakcja WW metodą pDM ... 148
13.6. Ekstrakcja WW metodą SM ... 152
13.7. Tworzenie słownika WW metodą SM ... 154
13.8. Ekstrakcja WW metodą SDM ... 156
13.9. Testowanie metod ekstrakcji WW ... 157
13.10. Podsumowanie ... 159

1. Wstęp

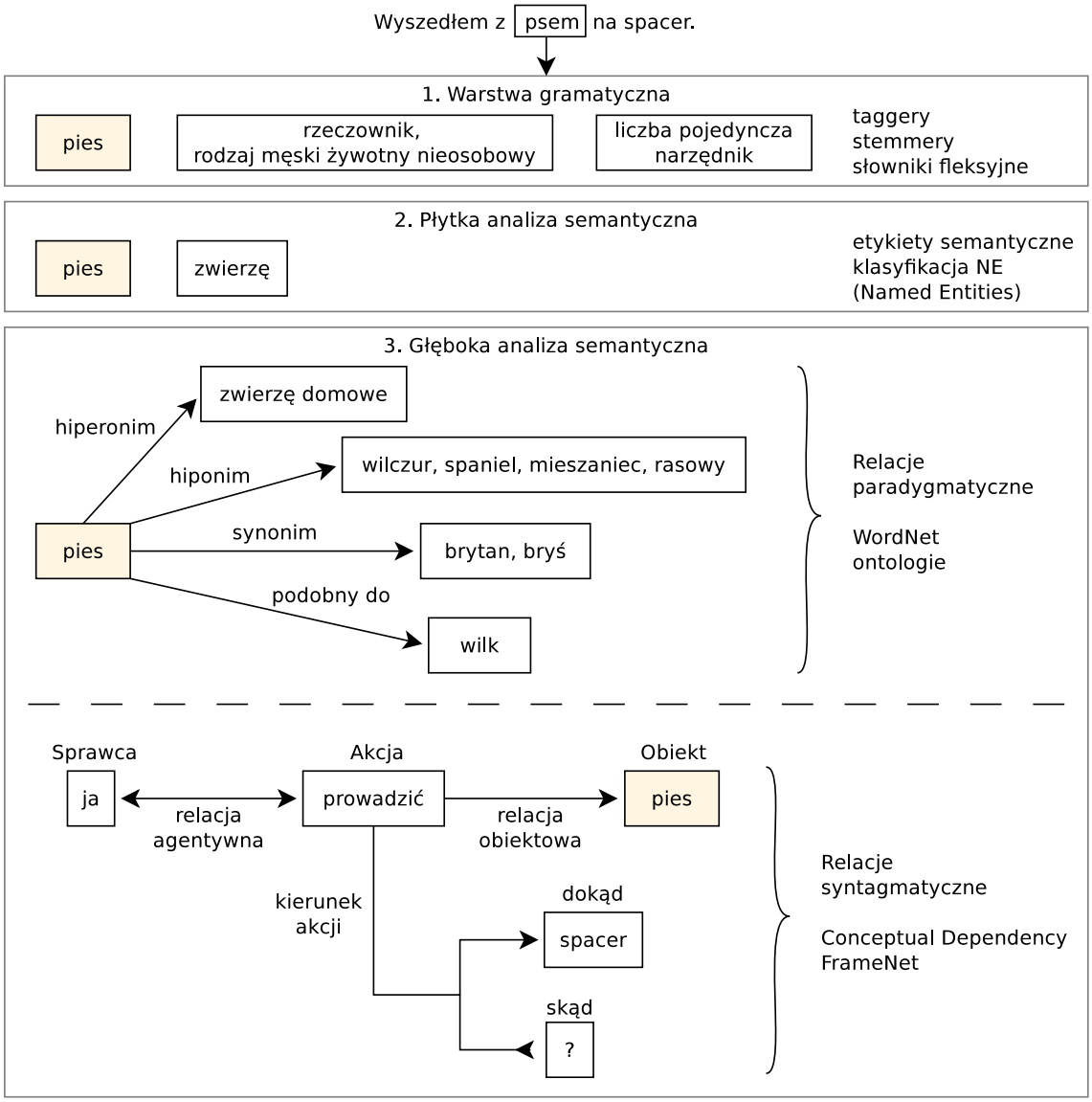
Przyjmuje się, że od momentu wynalezienia druku około 80% informacji jest przekazywanych i przechowywanych w postaci tekstu. Rozwój komputerów i Internetu sprawił, że ilość generowanych danych rośnie coraz szybciej. Powoduje to konieczność przetwarzania ich w sposób automatyczny np. dla celów wyszukiwania informacji czy też tłumaczenia tekstów. Obecnie wyszukiwarki oraz automatyczne serwisy tłumaczące tekst[1] posiadają nie tylko coraz większą funkcjonalność, ale też rosnącą jakość wyników.

Wysoką jakość przetwarzania tekstu w języku naturalnym można uzyskać jedynie przy użyciu zaawansowanych algorytmów i zasobów językowych. Metody statystyczne, takie jak np. wyszukiwanie dokumentów według statystyki częstości słów TF-IDF [79], mają ograniczoną skuteczność. Jest to spowodowane faktem, że operują one jedynie na liczbowych charakterystykach wyrazów, a nie na ich formie i znaczeniu. Aby uzyskać lepsze wyniki, trzeba wykorzystać cechy gramatyczne ekstrahowane z tekstu. W przypadku języka polskiego (i innych języków fleksyjnych) takimi cechami są: część mowy, forma podstawowa słowa, a także kategorie gramatyczne takie jak przypadek, liczba, rodzaj itp. Przykład ekstrakcji takich cech z tekstu przedstawiono na na rysunku 1.1 w warstwie nr 1. W analizowanym zdaniu wystąpiło słowo “psem”, dla którego rozpoznano formę podstawową “pies” oraz kategorie gramatyczne, zarówno te ustalone (część mowy, rodzaj) jak i zmienne (przypadek, liczba).

Ekstrakcja wyżej wymienionych cech może zostać wykonana przy użyciu różnych metod, jednak każda z nich wymaga pewnych zasobów językowych. Najprostszym, używanym od lat rodzajem takiego zasobu jest **słownik**, czyli baza danych zawierająca wyrazy danego języka z dodatkową informacją o nich. Zasoby słownikowe wykorzystywane są przez **analizatory morfologiczne** takie jak Morfeusz [107] i Morfologik [57]. Alternatywą dla tych narzędzi jest Słownik Fleksyjny Języka Polskiego (SFJP)[2]. Każdy z tych trzech zasobów zawiera ponad 100 tysięcy form podstawowych, pokrywając znaczącą większość wyrazów występujących w tekście polskim. Mimo to istnieje jednak potrzeba rozpoznawania słów występujących rzadko, których w słowniku nie ma – dlatego też istnieją narzędzia zwane **stemmerami** (wyznaczające formę podstawową słowa) i **tagerami** (tagujące, czyli oznaczające tekst cechami gramatycznymi). Narzędzia te wykorzystują zwykle algorytmy statystyczne takie jak np. SVM (*Support Vector Machine*) [30, 28], model maksymalnej entropii [81], HMM (*Hidden Markov Model*, ukryty

[1] np. serwis Google: wyszukiwarka i tłumacz, `http://www.google.com`

[2] SFJP – Słownik języka polskiego tworzony przez Grupę Lingwistyki Komputerowej w Katedrze Informatyki Akademii Górniczo-Hutniczej we współpracy z Katedrą Lingwistyki Komputerowej Uniwersytetu Jagiellońskiego [45].



Rysunek 1.1: Możliwe warstwy analizy tekstu na przykładzie zdania “Wyszedłem z psem na spacer”.

model Markowa) [27, 6] czy też CRF (*Conditional Random Field*) [37, 95, 4]. Są one trenowane na korpusie tekstu otagowanym wzorcowymi znacznikami. Pozwala to na tagowanie tekstu częściami mowy ze skutecznością 96-98%, także dla języka polskiego [39]. Ponadto wykorzystanie wielu metod jednocześnie wraz ze strategią głosującą poprawę skuteczności [40]. Dodatkowo ostatnio zaczęły powstawać wielowarstwowe tagery zoptymalizowane pod kątem dużego rozmiaru tagsetu dla języków fleksyjnych, jak np. WCRFT [74] czy Concraft [103], pozwalające na oznaczanie nie tylko części mowy, ale także innych cech fleksyjnych z dokładnością około 90%. Dobre wyniki uzyskiwane przez tagery i stemmery wynikają głównie z faktu, że rzadko występujące słowa posiadają zwykle typową strukturę gramatyczną i sposób odmiany – tworzone są one często według ustalonych reguł słowotwórczych np. dodanie sufiksu “-ować” do dowolnego rzeczownika przekształci go w poprawny gramatycznie czasownik, powiedzmy “czas” $\rightarrow$ “czasować”, który jednak nie zawsze będzie miał poprawnie zbudowane znaczenie. Oprócz

wspierania słowników fleksyjnych w zakresie słów nieznanych, tagery pozwalają ujednoznacznić niektóre słowa. Przykładowo, słowo "woli" może oznaczać rzeczownik "wola" w dopełniaczu lub przymiotnik (od: "wół"). Słownik jako baza danych nie pozwoli na rozstrzygnięcie tej niejednoznaczności, podczas gdy algorytm tagujący biorący pod uwagę kontekst wyrazu może tego dokonać.

Opisane cechy syntaktyczne znacząco rozszerzają możliwości automatycznego przetwarzania tekstu, jednak często okazują się one niewystarczające. Przykładowo, szukając w pewnym dokumencie zdań związanych z wyrazem "zwierzę" nie znajdziemy zdania "Wyszedłem z psem na spacer.". Aby było to możliwe, konieczne jest ekstrahowanie z tekstu informacji semantycznych, opisujących znaczenie słów. Warstwy 2 i 3 na rys. 1.1 przedstawiają możliwe modele danych semantycznych dotyczących segmentu "pies". W najprostszym przypadku dokonujemy jedynie płytkiej analizy polegającej wyłącznie na prostej klasyfikacji semantycznej – przykładowo słowo "pies" możemy zaliczyć do klasy "zwierzę", co pozwoli na rozwiązanie opisanego powyżej problemu wyszukiwania. Do płytkiej klasyfikacji semantycznej możemy też zaliczyć kategoryzację Named Entities (NE) – problem ten zostanie bliżej przedstawiony w podrozdziale 2.1. Bardziej rozbudowanym zasobem umożliwiającym głęboką klasyfikację semantyczną jest **sieć semantyczna**, wiążąca wyrazy różnymi **relacjami** (warstwa 3 na rys. 1.1). Istnieją dwa główne rodzaje takich sieci. Pierwszy z nich polega na utworzeniu klasyfikacji bazującej na **relacjach paradygmatycznych** takich jak synonimia, hiponimia i hiperonimia. Przykładem tego rodzaju sieci semantycznej jest WordNet [21], dla którego powstała polska wersja – SłowoSieć [49]. Bardziej rozbudowanym typem sieci są ontologie, np. CYC [48] dla j. angielskiego, które organizują wyrazy w skomplikowane taksonomie. Zasoby takie są niekompletne, a ich tworzenie wymaga ręcznego wysiłku.

Główną wadą sieci taksonomicznych jest jednak brak **relacji syntagmatycznych** [67], czyli zależności semantycznych obrazujących związki między wyrazami zachodzące w konkretnym zdaniu. W modelu Chomsky'ego [9] zależności syntagmatyczne (łączliwość semantyczna) pomiędzy wyrazami opisywano za pomocą cech selekcyjnych, np. czasownik "szczekać" wymusza, by wykonawcą czynności był pies, a przymiotnik "młody" może opisywać jedynie rzeczownik ożywiony. Pozwala to np. rozstrzygnąć, że zdanie "Azor szczekał, więc szybko wróciłem ze spaceru" również dotyczy psa (słowo "szczekać" jest czynnością powiązaną ze słowem "pies").

W teorii *conceptual dependency* wprowadzonej przez R. Schanka [92] definiuje się zależności pomiędzy **rolami semantycznymi**, np. *sprawca* akcji, *akcja*, *obiekt* akcji, *narzędzie* akcji itp., określającymi strukturę semantyczną zdania. Znaczenie wyrazu wchodzi w zależności syntagmatyczne dopiero wtedy, gdy wyraz zaczyna pełnić w zdaniu określoną rolę. W dolnej części rys. 1.1 przedstawiono przykład takiej struktury, która w analizowanym zdaniu wyróżnia podstawowe role semantyczne: sprawcę (podmiot "ja"), akcję ("prowadzić") i obiekt ("pies"), a także kierunek akcji (punkt początkowy jest nieznany, natomiast punktem docelowym jest spacer). Współczesnym przykładem systemu bazującego na relacjach syntagmatycznych jest FrameNet [85]. Opiera się on na koncepcji *frame semantics*: analizując zdania z korpusów wyróżniono około 800 **ram** (*frames*) – struktur językowych , które odpowiadają określonym sytuacjom semantycznym (np. gotowanie, sen, zemsta). Każda rama zawiera swoiste role semantyczne, np. *krzywda*, *skrzywdzony*, *krzywdziciel* i *mściciel* dla ramy *zemsta*, a konkretne jednostki leksykalne (wyraz +

znaczenie) pełnią w zdaniu określone role. Niestety dla języka polskiego podobny zasób semantyczny do tej pory nie powstał – trzeba podkreślić, że utworzenie tego rodzaju zasobów wynika dużego nakładu pracy. Obecnie powstaje również Słownik Semantyczny Języka Polskiego (SSJP) – semantyczna warstwa dla SFJP[3]. Zawiera on relacje syntagmatyczne, lecz niestety jest on jeszcze w dużym stopniu niekompletny, ponieważ tworzony jest ręcznie.

Niekompletność sieci semantycznych (zwłaszcza dla języka polskiego) uzasadnia obecność warstwy nr 2 na rys. 1.1 – o ile dla słów często występujących w języku można oczekiwać, że będą one obecne w wymienionych wyżej zasobach semantycznych, to rzadziej używane wyrazy najprawdopodobniej się w nich nie znajdą. W tym przypadku przydać się mogą automatycznie wyznaczone **etykiety semantyczne** – krótkie definicje umożliwiające identyfikację kategorii lub klasy, do której należy dany obiekt. Przykładem wykorzystania etykiet jest przetwarzanie wyrażenia "błąd pilota Cessny". Słowo "Cessna" jako nazwa własna najprawdopodobniej nie znajdzie się w sieci semantycznej, natomiast można przyporządkować mu etykietę "samolot", która znajdzie się już w takiej sieci. Ponieważ słowa "samolot" i "pilot" są powiązane semantycznie, będziemy mogli rozpoznać, że niejednoznaczne słowo "pilot" oznacza w tym przypadku człowieka, a nie np. rybę czy urządzenie sterujące. Ponadto jeżeli opisywany algorytm zostanie użyty w wyszukiwarce dokumentów, powyższe wyrażenie zostanie odnalezione np. dla frazy "lotnictwo", a nie dla "wędkarstwo".

Obecnie dostępne zasoby językowe dla języka polskiego zawierają głównie wyrazy jednosegmentowe – zwykle nie znajdziemy w nich wyrażeń składających się z kilku segmentów, w tym terminów ("czarna dziura"), idiomów ("panna młoda"), nazw własnych ("Polski Związek Wędkarski") czy też nazw osób ("Lech Wałęsa")[4]. Takie wyrażenia tworzą **wielosegmentowe języki leksykalne**, które w uproszczeniu nazywamy **wyrazami wielosegmentowymi** – będziemy również używać skrótu **WW**. Posiadają one własne znaczenie, często inne niż suma znaczeń poszczególnych segmentów [44], więc jeżeli np. wyrażenie "panna młoda" występujące w wyszukiwanym dokumencie nie znajdzie się w słowniku, nie będzie możliwe rozpoznanie, że dokument dotyczy ślubu, ponieważ same wyrazy "panna" i "młoda" nie są semantycznie powiązane ze ślubem. Ponadto nierozpoznane nazwy własne mogą zostać błędnie zinterpretowane jako sekwencje nie powiązanych ze sobą wyrazów. Powoduje to konieczność dołączenia informacji dotyczących wyrazów wielosegmentowych do zasobów językowych: słowników fleksyjnych, słowników semantycznych i otagowanych korpusów.

Powyższe rozważania prowadzą do wniosku, że potrzebne jest uzupełnienie zasobów językowych dla języka polskiego, w szczególności w zakresie etykiet semantycznych i wyrazów wielosegmentowych – dlatego też na tym właśnie koncentrują się badania opisane w niniejszej pracy. W tab. 1.1 przedstawiono podsumowanie narzędzi służących do przetwarzania języka polskiego. Kursywą wyróżniono narzędzia, które są przedmiotem przedstawionych w niniejszej pracy badań.

[3]Tworzony jest on w Katedrze Lingwistyki Komputerowej UJ we współpracy z Grupą Lingwistyki Komputerowej Katedry Informatyki AGH.

[4]W ostatnich latach powstają jednak zasoby je zawierające, np. NELexicon [46] zawierający nazwy oraz polski WordNet zawierający niektóre wyrazy pospolite. Zasoby te rozwijane są równolegle z badaniami opisywanymi w niniejszej pracy i w dalszym ciągu są niekompletne – zwłaszcza pod względem opisu struktury wyrazów.

Tabela 1.1: Narzędzia umożliwiające przetwarzanie tekstu w języku polskim. Obszary pokrywające się z zakresem opisywanych badań oznaczono *kursywą*. Badania nad ekstrakcją wyrazów wielosegmentowych oznaczono kolorem zielonym, a na pomarańczowo zaznaczono badania nad ekstrakcją etykiet semantycznych dla haseł Wikipedii.

	Wyrazy jednosegmentowe		**Wyrazy wielosegmentowe (WW)**	
	częste	rzadkie	częste	rzadkie
Syntaktyka	Słowniki i analizatory morfologiczne, np. SFJP, Morfologik, Morfeusz	Stemmery / tagery	*Słownik wyrazów wielosegmentowych*	*Metody ekstrakcji wyrazów wielosegmentowych*
Semantyka	ontologie, WordNet, SSJP	*Etykiety semantyczne*	*Etykiety semantyczne*	*Metody ekstrakcji etykiet semantycznych z tekstu*

W pierwszej kolejności zaprojektowano i zaimplementowano algorytm ekstrakcji etykiet semantycznych dla haseł Wikipedii (jedno- i wielosegmentowych). Główna część tego algorytmu powstała już w ramach pracy magisterskiej Autora [10], natomiast później metoda ta została jeszcze uzupełniona [11] i dostosowana do nowego formatu bazy danych. Obszar ten w tabeli 1.1 oznaczono na pomarańczowo. Głównym przedmiotem niniejszej pracy jest z kolei uzupełnienie zasobów językowych o informacje morfosyntaktyczne dla wyrazów wielosegmentowych. W tab. 1.1 obszar ten oznaczono kolorem zielonym. Podjęto decyzję o wykorzystaniu polskiej Wikipedii [105] jako źródła WW. Początkowe efekty opisano w publikacji [12], natomiast niniejsza praca opisuje ostateczną wersję opracowanych algorytmów.

1.1. Tezy pracy

Podstawowym celem niniejszej pracy jest umożliwienie ekstrakcji wyrazów wielosegmentowych dla języka polskiego – pozwala to sformułować pierwszą tezę pracy.

TEZA 1

Możliwe jest opracowanie algorytmu ekstrahującego w sposób automatyczny wyrazy wielosegmentowe z tekstu w języku polskim, wykorzystującego jako źródła danych słownik fleksyjny i Wikipedię.

Algorytm ekstrakcji może działać samodzielnie, jednak przede wszystkim może on zostać użyty do stworzenia słownika WW. Dlatego też w niniejszej pracy wykazana zostanie również prawdziwość poniższej tezy.

TEZA 2

Możliwe jest utworzenie w sposób automatyczny słownika wyrazów wielosegmentowych z ha-

seł Wikipedii oraz wyrazów wielosegmentowych wyekstrahowanych przy pomocy algorytmu opisanego w Tezie 1.

Niniejsza praca opisuje więc głównie badania nad **ekstrakcją** wyrazów wielosegmentowych. Odnośnie **klasyfikacji semantycznej** tych wyrazów, prace ograniczono do dopracowania wcześniejszego algorytmu wyznaczającego etykiety semantyczne haseł Wikipedii [11] oraz wstępnych eksperymentów dotyczących wyznaczania takich etykiet dla nowo wyekstrahowanych wyrazów. Dalsze badania ujęte są w planach przyszłych prac.

1.2. Struktura pracy

W rozdziale 2 przedstawiono bieżący stan badań. Skupiono się przede wszystkim na ekstrakcji WW, a także przedstawiono wyniki badań nad ekstrakcją informacji z Wikipedii. Następnie w rozdziale 3 opisano opracowane na potrzeby niniejszej pracy narzędzia słownikowe. W rozdziale 4 zaprezentowano z kolei proces ekstrakcji danych z Wikipedii. Rozdział 5 poświęcono ekstrakcji etykiet semantycznych z haseł Wikipedii – ponieważ sam algorytm został już opisany w pracy magisterskiej Autora [10], a jego ulepszona wersja została opublikowana [11], opis skrócono do minimum i skupiono się na różnicach w stosunku do dotychczasowej wersji algorytmu.

Zasadniczy problem ekstrakcji wyrazów wielosegmentowych zdefiniowano w rozdziale 6. Kolejne rozdziały prezentują metody rozwiązania tego problemu. Rozdział 7 przedstawia metodę DM – najprostsze rozwiązanie problemu, wykorzystujące Wikipedię jako słownik WW. Jego ulepszoną wersję, pDM, zaprezentowano w rozdziale 8. Wykorzystuje ona wyznaczone dla haseł Wikipedii wzorce odmiany w celu rozpoznawania segmentów odmiennych. Ponieważ metoda ta rozpoznaje wyłącznie hasła Wikipedii, istnieje potrzeba opracowania algorytmu, który potrafi ekstrahować z tekstu nowe wyrazy o strukturze zbliżonej do znanych już WW – tę metodę, nazwaną SM, opisuje rozdział 9. Rozdział 10 prezentuje metodę SDM wykorzystującą słownik WW utworzony przy pomocy metody SM oraz metodę łączoną, osiągającą najlepsze wyniki. Przedstawione metody wykazują prawdziwość Tezy 1 (skutecznie ekstrahują WW) oraz Tezy 2 (tworzony jest słownik WW zawierający hasła Wikipedii oraz wyrazy uzyskane poprzez ekstrakcję WW metodą SM).

Możliwości dalszego rozwoju opisanych metod, a także wyznaczania etykiet semantycznych dla nowo wyekstrahowanych wyrazów zaprezentowano w rozdziale 11. Rozdział 12 podsumowuje całą pracę.

2. Stan badań

Ekstrakcja wyrażeń wielosegmentowych z tekstu jest zagadnieniem dość szeroko opisywanym zwłaszcza dla najbardziej rozpowszechnionych języków takich jak angielski i francuski. W literaturze anglojęzycznej funkcjonuje pojęcie *Multi-Word Expression* (MWE)[1]. Termin ten został zdefiniowany w pracy Saga i in. [86] w bardzo ogólny sposób jako "idiosynkratyczne interpretacje przekraczające granice słów". Problem przetwarzania MWE jako odrębnych jednostek leksykalnych jest złożony, dlatego zwykle rozważa się jego szczególny przypadek – Named Entity Recognition.

2.1. Named Entity Recognition (NER)

Istnieje gałąź przetwarzania MWE, która została już dość dokładnie zbadana – jest to **Named Entity** (NE) **Recognition** (NER). Zagadnienie to polega na ekstrakcji z tekstu jednostek nazewniczych należących do określonych kategorii takich jak osoby ("Jan III Sobieski", "premier Polski"), obszary geograficzne ("Wyżyna Krakowsko-Częstochowska", "stan Michigan"), nazwy instytucji ("Polski Związek Łowiecki"), a także np. wyrażenia określające datę ("19 stycznia 1978") czy ilość pieniędzy w pewnej walucie ("189 złotych 40 groszy"). Cechą wspólną tych wyrażeń jest duża zależność między strukturą syntaktyczną a kategorią semantyczną, do której należą. Typowe algorytmy ekstrakcji NE wykorzystują klasyfikatory i tagery trenowane na podstawie korpusów, w których poszczególne segmenty zostały oznaczone kategoriami NE, do których należą. Wykorzystywane metody klasyfikacji są zwykle podobne jak dla tagowania częściami mowy. Skuteczność algorytmów opisuje się zwykle podając wartości miar *precision* (**precyzja**, czyli odsetek poprawnych wyników wśród rezultatów rozpoznawania), *recall* (**pełność**, czyli odsetek rozpoznanych wyników spośród wszystkich NE występujących w próbce testowej) i F_1 (F-measure, czyli średnia harmoniczna precyzji i pełności). Zagadnienie jest na tyle szeroko znane, że odbyły się konferencje, na których tematem przewodnim były zadania polegające na rozwiązywaniu tego samego problemu NER przez wielu autorów (tzw. *shared task*). Przykładem może być praca [97], opisująca zadanie z konferencji CoNLL[2] z roku 2003. Problem polegał na ekstrakcji czterech kategorii NE (nazw osób, lokalizacji, organizacji i innych) z tekstu w języku angielskim i niemieckim. Dostępne były treningowe i testowe próbki ręcznie otagowanego tekstu dla obu języków. Autorzy poszczególnych rozwiązań stosowali podobne podejście – poszczególne segmenty próbki treningowej służyły do ekstrakcji

[1] Czasem spotyka się też pojęcie *Multi-Word Unit* (MWU).

[2] Conference on Computational Natural Language Learning

cech, takich jak np. n-gramy (n-ki liter), przed/przyrostki, kategorie gramatyczne, informacja o przynależności do grup składniowych (np. wyrażenia rzeczownikowego), interpunkcja, pisownia wielką/małą literą itp. Następnie wyekstrahowane cechy wraz z tagami NE dla zbioru treningowego służyły do dobrania optymalnych parametrów wybranego klasyfikatora tak, by był on w stanie tagować jak najdokładniej próbkę treningową, a poprzez generalizację również próbkę testową.

Przykładowe rozwiązanie tego zadania opisane w pracy [23] pokazuje, że dobre wyniki uzyskać można łącząc rezultaty kilku typowych modeli statystycznych. Autorzy wykorzystali m.in. metodę RRM (*Robust Risk Minimization*) – probabilistyczny model opierający się na minimalizacji funkcji ryzyka [110]. Kolejna użyta metoda to model maksymalnej entropii [81], czyli model statystyczny, w którym podczas treningu próbujemy znaleźć klasyfikację o największej wartości funkcji entropii. Ponadto wykorzystano ukryty model Markowa (HMM, *Hidden Markov Model*) [27], czyli model sekwencyjny, który zakłada zależność tagu bieżącego segmentu od ukrytego stanu zmieniającego się z każdym przeczytanym segmentem. W końcu użyto też narzędzia fnTBL [24], wykorzystującego opartą na regułach metodę *transformation-based learning* wprowadzoną przez Brilla [7]. Połączenie tych metod dało w efekcie precyzję, pełność i wartość F_1 równą 89% dla języka angielskiego oraz odpowiednio precyzję 84%, pełność 64% i wskaźnik F_1 równy 72% dla niemieckiego. Niska wartość pełności dla j. niemieckiego tłumaczona była głównie gorszą jakością leksykonów, jednak z pewnością nie bez znaczenia była też deklinacja, która nie występuje w języku angielskim[3].

Opisane rozwiązanie było to najlepszym spośród opisanych w pracy [97], jednak warto wspomnieć, że jeszcze wyższy wynik dała metoda łącząca wyniki kilku najlepszych prac poprzez strategię głosowania – jest to na pewno dobre podejście, pozwalające na wyeliminowanie wad poszczególnych metod składowych, które zwykle nie są powtarzalne. Warunkiem jest jednak użycie dostatecznie odmiennych metod składowych o wysokiej jakości. Wśród innych podejść warto wymienić pracę [50], w której użyto metody *linear-chain* CRF (*Conditional Random Field*), którą w pewnym stopniu możemy uważać za dyskryminatywne (warunkowe) rozwinięcie HMM lub sekwencyjny odpowiednik modelu maksymalnej entropii [95]. W pracy tej pokazano również, że ważne jest wykorzystanie dobrych leksykonów wspomagających użyty tager – w tym przypadku użyto zasobów wyekstrahowanych z różnych stron internetowych. Ponadto ważną rolę w NER pełni ekstrakcja właściwych cech z tekstu. Przekonuje o tym praca [112], w której użycie starannie dobranych cech pozwoliło na osiągnięcie wysokiego wyniku. Co więcej, wynik ten poprawiono wykorzystując informację o tym, jakie etykiety przydzielono danemu kandydatowi na NE w dotychczas przetworzonej części tekstu. Na koniec uzyskano F_1 przekraczające 94% dla próbki w języku angielskim[4].

Metody NER są stale rozwijane, co pozwala na podnoszenie wyników dla znanych zbiorów – np. w pracy [80] pokazano, jak m.in. wzbogacenie języka znaczników określających przynależność segmentu do danego NE oraz poprawa wykorzystywanych cech pozwala na uzyskanie rekordowej wartości F_1 zbliżonej do 91% dla języka angielskiego na zbiorze CoNLL-2003. Wśród zastosowanych nowych

[3] Dla języka polskiego fleksja jest jeszcze bardziej rozbudowana, więc można by się spodziewać znacznego zwiększenia tych problemów.

[4] Użyty zestaw testowy (MUC-7) był jednak inny niż w pozostałych pracach (CoNLL-2003).

metod generowania cech warto wspomnieć wykorzystanie informacji o szerszym kontekście danego segmentu oraz klasteryzacji segmentów w postaci drzewa – ta druga modyfikacja pokazuje że uczenie bez nadzoru (*unsupervised learning*) może poprawić wyniki bazujące na klasycznym uczeniu maszynowym z nadzorem (*supervised learning*).

W ostatnich kilku latach popularne stają się metody, które wykorzystują uczenie bez nadzoru w coraz większym stopniu. W pracy [19] opisano system, który wykorzystuje technikę *bootstrappingu* do automatycznego tworzenia leksykonów. Metoda ta polega na wykorzystaniu kilku przykładowych NE należących do danej klasy (np. "New York" i "Los Angeles" dla klasy "city") do skonstruowania reguł określających konteksty, w których miasto to może wystąpić. Reguły tworzone są z ręcznie utworzonych wzorców, np. wzorzec "*«class»* such as *«noun phrase»*" dla klasy "city" powoduje utworzenie m.in. reguły "cities such as *«noun phrase»*". Utworzone reguły są następnie używane do wyszukania dokumentów w sieci WWW. Zwracane wyniki używane są do stwierdzenia, czy poszczególne wyrażenia pasujące do reguły należą do danej klasy. W tym celu używana jest miara PMI-IR (*pointwise mutual information* [100] – jedna z miar współwystępowania wyrażeń), oceniająca współwystępowanie np. wyrażenia "Atlanta" i całej frazy wyszukiwania "cities such as Atlanta". Ponieważ reguł dla danej klasy jest wiele, wykorzystywany jest dodatkowo klasyfikator *Naive Bayes* [32], który na podstawie dostarczonych pozytywnych i negatywnych przykładów podejmuje ostateczną decyzję o dołączeniu danego wyrazu do leksykonu. Rozwinięciem powyższych idei jest praca [61], w której wyszukiwane strony internetowe są dodatkowo analizowane pod kątem wystąpienia charakterystycznych znaczników (np. hiperłącza, pogrubiony tekst), a następnie wyrażenia występujące w tych znacznikach klasyfikowane są jako należące (bądź nie) do wynikowego leksykonu. Autorzy tworzą kompletny system NER bazujący na utworzonych leksykonach i dodatkowych heurystycznych algorytmach ujednoznaczniających – uzyskana wartość F_1 wynosząca 88% pokazuje, że system ten może konkurować z typowymi algorytmami stosującymi uczenie z nadzorem. Potwierdzenie zmiany tendencji w algorytmach NER poprzez coraz częstsze stosowanie algorytmów uczenia bez nadzoru można również znaleźć w artykule [60].

W wymienionych wyżej pracach można znaleźć wartości F_1 wynoszące 70-95% na różnych korpusach, jednak należy podkreślić, że badania te ograniczone są do starannie wybranych kategorii NE (zwykle tych dających najlepsze wyniki), a algorytmy wykorzystują często ręcznie tworzone reguły i leksykony. Cel ekstrakcji WW opisywanej w niniejszej pracy jest odmienny – po pierwsze chcielibyśmy ekstrahować z tekstu nie tylko Named Entities, ale także pospolite WW. Poza tym naszym celem jest ekstrahowanie dowolnego rodzaju wyrazów wielosegmentowych, bez ograniczeń i kategoryzacji. Dlatego też nie powinno się wprowadzać żadnych odgórnych reguł, które mogłyby zawęzić obszar poszukiwań. Na obecnym etapie nie można więc wykorzystać typowych algorytmów NER. Opisane powyżej metody uczenia bez nadzoru [19, 61] w pewnym stopniu pokrywają się jednak z problemem ekstrakcji WW. Z całą pewnością użyty pomysł przeszukiwania Sieci w selu rozbudowy leksykonu jest dobrym pomysłem, celem pracy jest jednak ekstrakcja WW z dowolnego tekstu – nie możemy ograniczyć się więc jedynie do stworzenia leksykonu. Ponadto wydaje się, że użyte techniki najlepiej sprawdziłyby się do rozbudowy już istniejących zasobów, ponieważ crawling bazujący na frazach wyszukiwania zawsze obarczony jest

pewnym ograniczeniem precyzji płynącym z niskiej jakości większości dokumentów znajdujących się w sieci Internet. Problemem jest też konieczność tworzenia heurystycznych reguł wykorzystanych w powyższych algorytmach – w niniejszej pracy tego właśnie chcielibyśmy uniknąć. W badaniach wykorzystano natomiast ideę rozpoznawania wzorców pozwalających na lokalizowanie wyrazów wielosegmetowych, przy czym wzorce te tworzone są w pełni automatycznie.

2.1.1. NER dla języka polskiego

W przypadku języka polskiego głównym problemem jest duża liczba możliwych form gramatycznych i złożone reguły odmiany wyrazów, które w przypadku NE stają się jeszcze bardziej skomplikowane ze względu na różne zależności między segmentami. Powoduje to zmniejszenie możliwości zastosowań typowych metod statystycznych i konieczność stosowania systemów opartych o reguły heurystyczne. W pracach [66, 65] wykorzystano narzędzie SProUT, które rozpoznaje NE opierając się na ręcznie utworzonych regułach gramatycznych i dodatkowym leksykonie. Do rozpoznawania różnych form gramatycznych pojedynczych słów użyto analizatora morfologicznego Morfeusz [107]. Test został przeprowadzony na próbce 100 wiadomości finansowych z gazety Rzeczpospolita. Osiągnięto różne wyniki w zależności od kategorii NE – przykładowo wartości precyzji i pełności dla wyrażeń określających ilość pieniędzy to 98% i 94%, ale już dla nazw organizacji były one dużo niższe – 88% i 57%.

Inny system regułowy, mogący posłużyć m.in. do ekstrakcji NE dla języka polskiego to parser Spejd [8]. Służy on do częściowego parsingu, czyli identyfikacji słów składniowych (np. “polsko-japoński”) oraz grup składniowych, takich jak np. grupa nominalna (“Kościół św. Krzyża”) czy też wyrażenie przyimkowe (“w Europie”). Narzędzie to na wejściu przyjmuje korpus tekstu w formacie XML otagowany znacznikami morfosyntaktycznymi (generowanymi np. przez Morfeusza), natomiast na wyjściu otrzymujemy ten sam korpus wzbogacony o dodatkowe znaczniki oznaczające rozpoznane wyrażenia. Dodatkowo narzędzie to pozwala na ujednoznacznienie niektórych znaczników – przykładowo w wyrażeniu “z drzewa” słowo “drzewa” mogłoby wystąpić w mianowniku liczby mnogiej, lecz rozpoznając wyrażenie przyimkowe odrzucimy tę możliwość.

Co prawda Spejd wydaje się być narzędziem mogącym jedynie wspomagać NER w zakresie wstępnego rozpoznawania i ujednoznaczniania grup składniowych, jednak użyty język reguł został z powodzeniem wykorzystany w pracy [91] do opisu specjalistycznego leksykonu terminów ekonomicznych SEJFEK. Leksykon ten liczy ponad 11 tys. haseł, które zostały ręcznie wyodrębnione z różnych zasobów. Do manualnego opisu sposobu odmiany poszczególnych segmentów użyto narzędzia Toposław [47]. Jest to dość rozbudowane i funkcjonalne narzędzie pozwalające na tworzenie słowników wyrażeń wielosegmentowych. W pierwszej kolejności wykorzystuje ono Morfeusza do morfosyntaktycznego znakowania segmentów. Następnie użytkownik przy pomocy interfejsu graficznego dokonuje ujednoznaczniania segmentów i wybiera segmenty odmienne. Kolejny krok polega na konstruowaniu na podstawie utworzonego opisu grafu odmiany. Graf ten wykorzystywany jest do wygenerowania odmienionych form. Ostatnią operację umożliwia użyty generator form fleksyjnych Multiflex [88]. Należy wspomnieć, że narzędzie to wspiera zarówno skróty (“Spółka Akcyjna” $\rightarrow$ “S.A.”), jak i nazwy złożone (“Bank BPH Spółka Akcyjna” zawiera

nazwę "Spółka Akcyjna", więc posiada wariant "Bank BPH S.A."). Powstały z użyciem Toposława leksykon został półautomatycznie przetworzony do postaci gramatyk systemu Spejd – okazuje się więc, że parser ten może równie dobrze służyć za słownik wyrazów wielosegmentowych. Użyte w nim reguły dodatkowo potrafią weryfikować poprawność wzorców odmiany wyrazów z leksykonu. Nie wiadomo jednak, jak narzędzie to skalowałoby się do bardzo dużych słowników – autorzy podkreślają, że szybkość działania jest mocno zależna od liczby i złożoności użytych reguł [8].

Powyższy leksykon umożliwia dokładne rozpoznawanie NE, lecz jest ograniczony do konkretnej dziedziny. W ostatnich latach, już podczas trwania badań nad niniejszą pracą, zaczęły jednak pojawiać się pierwsze ogólne zasoby NE dla języka polskiego. Duże znaczenie ma tutaj intensywnie rozwijany Narodowy Korpus Języka Polskiego (NKJP) [71]. Jest to duży liczący ponad miliard segmentów korpus, który został oznaczony na kilku różnych warstwach – pierwszą z nich jest tagowanie morfosyntaktyczne, wykonane głównie przy pomocy Morfeusza [107]. Korpus podzielono na część znakowaną ręcznie liczącą milion segmentów oraz pozostałą część, dla której operacje te wykonano automatycznie. Przykładowo, ujednoznacznianie niejednoznacznych cech morfosyntaktycznych dla pierwszej z tych części wykonano ręcznie, a do tagowania pozostałej części użyto narzędzia Pantera [2], czyli zoptymalizowanego dla języków fleksyjnych regułowego tagera Brilla [7]. Kolejne warstwy znaczników w NKJP to słowa składniowe i grupy składniowe. Tutaj wykorzystywano narzędzie Spejd [8]. Najbardziej interesująca w kontekście NER jest kolejna warstwa, obejmująca jednostki nazewnicze (NE) [90, 104, 89]. Dla liczącego milion segmentów podkorpusu została ona utworzona półautomatycznie. W pierwszej kolejności skompletowano leksykon i reguły dla opisywanego powyżej narzędzia regułowego SProUT. Użyty leksykon łączy w sobie oryginalny zasób stworzony dla narzędzia SProUT [65] oraz dodatkowe dane ze źródeł takich jak np. Wikipedia czy też indeks polskich nazw geograficznych – łącznie ponad 54 tys. jednostek leksykalnych. Utworzone reguły gramatyczne (jest ich ok. 160) są również rozwinięciem oryginalnych reguł narzędzia SProUT. Wysiłek włożony w rozwój opisanych zasobów dał w efekcie precyzję wynoszącą 72% i pełność 36% [89]. Rozpoznawane kategorie NE to nazwy osób, miejsc, organizacji i wyrażeń oznaczających czas, a także utworzone z nich wyrazy pochodne (np. "poznański", "Bawarczyk"). Wyniki były następnie poprawiane ręcznie przy użyciu edytora TrEd [63] przez dwóch językoznawców – został zastosowany dość złożony system uzgadniania rozbieżności. Oznaczony w ten sposób podkorpus o wielkości miliona segmentów zawiera 87300 jednostek nazewniczych i posłużył on następnie do wytrenowania tagera opartego na CRF, który został zastosowany dla pozostałego korpusu. Tager ten osiągnął wartość F_1 wynoszącą 79%. Należy podkreślić, że w powyższych badaniach uwzględniono m.in. rozpoznawanie jednostek zagnieżdżonych (np. jednostka "ul. kardynała Stefana Wyszyńskiego" zawiera m.in. jednostkę "Stefan Wyszyński"), a także struktury z nakładającymi się elementami (w wyrażeniu "z województw: poznańskiego i bydgoskiego" znajdują się jednostki "województwo poznańskie" i "województwo bydgoskie").

Na podstawie analizy powyższych artykułów możemy zaobserwować silny wpływ fleksji języka polskiego na używane algorytmy NER. Duża część prac korzysta z reguł gramatycznych i wydaje się, że jest to również metoda, którą trzeba stosować w niniejszej pracy do ekstrakcji WW. Dążenie do ekstrakcji jak największego zakresu WW prowadzi jednak do odrzucenia metod używających heurystycznie tworzone

wzorce składniowe – chcielibyśmy wzorce takie tworzyć w sposób automatyczny. Oczywiście narzędzia takie jak zasoby NKJP, a także platforma SProUT i parser Spejd mogą okazać się przydatne, zdecydowano jednak nie wykorzystywać ich na bieżącym etapie pracy, ponieważ używana byłaby tylko niewielka część ich funkcjonalności, a wykorzystane narzędzia słownikowe są z nimi niekompatybilne pod względem formatu danych leksykalnych (patrz rozdział 3). W późniejszych badaniach z całą pewnością należy jednak rozważyć użycie tych narzędzi, jak też i ręcznie utworzonych reguł gramatycznych – jest to dobra metoda dalszej poprawy wyników.

2.2. Rozpoznawanie i ekstrakcja wyrażeń wielosegmentowych (MWE)

Problemem bardziej ogólnym od NER jest rozpoznawanie MWE w tekście. Czasami też rozważa się uproszczoną wersję problemu, polegającą jedynie na zdecydowaniu, czy dane wyrażenie jest MWE (klasyfikacja binarna). Problem ekstrakcji MWE polega z kolei na tworzeniu leksykonów zawierających te wyrażenia z pewnych danych źródłowych (zwykle zbiorów tekstów).

Najprostsze metody wykrywania wystąpień MWE w tekście polegają na używaniu statystycznych miar współwystępowania słów. Przykładem jest praca [77], w której przeanalizowano przydatność kilku miar statystycznych do rozwiązywania problemu klasyfikacji binarnej MWE dla języka angielskiego i niemieckiego, przy czym badania ograniczono do zbiorów wyrażeń prostych strukturalnie (np. przymiotnik-rzeczownik i czasownik-przyimek). Zbadano m.in. miary Mutual Information (MI) i χ^2, które określają współwystępowanie słów w bigramach, a także wprowadzoną w pracy [111] miarę Permutation Entropy, która dodatkowo bierze pod uwagę kolejność wystąpień. Wyniki były niskie – uzyskano precyzję około 30-40% dla pełności 40-50%. Ponadto podjęto próbę wprowadzenia nowej miary EPI (*Entropy of Permutation and Insertion*). Bierze ona pod uwagę informacje o języku, rozpoznając dla każdego MWE jego gramatyczne warianty. Niestety również ta miara ta nie przyniosła poprawy wyników. Autorzy sugerują, że wyniki mogłaby poprawić kombinacja wielu miar i tezę tę potwierdza praca [64]. Połączono w niej aż 55 różnych miar statystycznych, tworząc wielowymiarowy wektor cech. Następnie na ręcznie oznaczonych zbiorach treningowych wytrenowano klasyfikatory statystyczne, które dokonywały ostatecznej decyzji. Użyto m.in. regresji logistycznej, dyskryminanty LDA oraz sieci neuronowych. Zarówno zbiory treningowe, jak i testowe zawierały jedynie bigramy o dobrze określonej strukturze (np. przymiotnik-rzeczownik), głównie dla języka niemieckiego. Ponieważ problem polegał wyłącznie na klasyfikacji binarnej, zamiast miar precyzji i pełności podano jedynie wartość dokładności wyników – dla wyrażeń rzeczownikowych była ona dość wysoka, ponad 60%, lecz dla wyrażeń czasownikowych dokładność mocno spadała. Autorzy dodatkowo sygnalizują problemy z jednoznaczną klasyfikacją wyrażeń jako MWE, wprowadzając dodatkową klasę słów o niepewnej przynależności.

Na powyższej koncepcji łączenia miar statystycznych i klasyfikatorów opiera się system MWEToolkit [78], służący do ekstrakcji MWE z tekstu. System ten w pierwszej kolejności dokonuje ekstrakcji n-ek segmentów pasujących do prostych wzorców syntaktycznych (np. przymiotnik-rzeczownik-rzeczownik), a następnie oblicza kilka miar statystycznych dla znalezionych kandydatów. W kolejnym kroku miary

te wykorzystywane są przez system uczenia z nadzorem. Opisany eksperyment polegał na ekstrakcji terminów biomedycznych z korpusu Genia [34], natomiast do ostatecznej klasyfikacji użyto klasyfikatora SVM [30]. Uzyskano niską wartość F_1 wynoszącą około 30%, jednak ciężko wynik ten odnieść do ogólnego problemu ekstrakcji MWE – był on przeprowadzony jedynie na ograniczonym zasobie językowym. MWEToolkit był też używany przez innych autorów, np w pracy [113] ekstrahując MWE z korpusu Europarl [36] uzyskano wysoką dokładność walidacji kandydatów na MWE – wartość F_1 wyniosła 84%. Autorzy nie zbadali jednak skuteczności późniejszego rozpoznawania MWE w tekście. Obiecujące wyniki dają metody wykorzystujące ręcznie utworzone zbiory danych treningowych, np. w pracy [20] wykorzystano zbiór 10 tys. ręcznie otagowanych pozytywnych i negatywnych próbek dla j. angielskiego do wytrenowania klasyfikatora SVM. Szczególny nacisk położono na cechy opisujące kontekst wystąpień, np. MWE często będą występowały otoczone parą słów "the (...) of". Uzyskana dokładność klasyfikacji binarnej wyniosła 89%, lecz niestety nie przeprowadzono pełnego testu ekstrakcji z tekstu, chociaż właśnie taki był nadrzędny cel autorów – podobnie jak w pracy [78] stworzono kompletny system ekstrakcji.

Dobre wyniki ekstrakcji MWE z tekstu osiągane są dla niektórych popularnych języków (angielski, francuski), dla których istnieją słowniki maszynowe i korpusy zawierające MWE. Przykładowo, dla języka francuskiego istnieje korpus French TreeBank [1], który zawiera otagowane MWE. Wykorzystano go w pracy [14] do wytrenowania tagera opartego na CRF, który znakuje segmenty zarówno częściami mowy, jak i przynależnością do MWE. Autorzy do tagowania użyli wielu cech, w tym cech pojedynczego segmentu (pisownia, przedrostki, przyrostki itp.), cech bigramów (tag poprzedniego segmentu), cech kontekstu, a także etykiet przydzielanych segmentom przez zewnętrzny słownik fleksyjny. Dodatkowo do oznaczania MWE wykorzystano zewnętrzne leksykony – dla języka francuskiego istnieją słowniki zawierające MWE takie jak DELAC [16] i Lefff [87]. Fakt rozpoznania MWE przez jeden z tych słowników jest oznaczany jako kolejna cecha wykorzystywana w tagerze CRF. Podczas testu tagera uzyskano skuteczność tagowania segmentów 94.4%, co w przypadku rozpoznawania MWE przekłada się to na wartość F_1 w granicach 70-80%. Autorzy łącząc swoją metodę rozpoznawania MWE z parsingiem [15] uzyskali dalszą poprawę wyników.

Można zauważyć, że we wszystkich powyższych pracach dających dobre wyniki [14, 15, 20] wykorzystywano uczenie z nadzorem (*supervised learning*), wymagające dobrej jakości zbiorów treningowych, a dodatkowo przydatne były słowniki zawierające MWE. W przypadku języka polskiego póki co nie istnieją słowniki fleksyjne zawierające MWE – jedynie powstający równolegle z niniejszą pracą polski WordNet [49] zawiera już niektóre WW (nie ma tam jednak informacji fleksyjnej), natomiast korpus NKJP otagowany jest tylko kategoriami NE. Dlatego wydaje się, że w przypadku języka polskiego krok polegający na utworzeniu zasobów zawierających WW należy dopiero wykonać – jest to jeden z celów niniejszej pracy. Chcielibyśmy uniknąć ekstrakcji WW według ręcznie tworzonych reguł (ogranicza to w sposób sztuczny zakres ekstrakcji), a ponieważ użycie pewnych zasobów językowych jest konieczne, naturalnym wyborem jest **Wikipedia**, wykorzystana już wcześniej przez Autora do ekstrakcji etykiet semantycznych [10, 11]. Jest to zasób w pewnym stopniu ustrukturyzowany, a z drugiej strony nie jest to

typowy leksykon MWE – unikalny charakter tego zasobu językowego powoduje, że warto przyjrzeć się dokładniej dotychczasowym badaniom wykorzystującym tę encyklopedię – opisano je w podrozdziale 2.3.

2.2.1. Struktura MWE w języku polskim

W przypadku języka polskiego autorzy niejednokrotnie próbują znaleźć sposób opisu MWE odpowiadający szczególnym cechom tego języka. Autor pracy [58] rozważa, w jaki sposób połączyć podejście oparte na parsingu z rozpoznawaniem MWE, postulując istnienie dwóch klas takich wyrażeń. Pierwsza z nich obejmuje wyrażenia morfologicznie idiosynkratyczne, czyli takie, których szczególne cechy przejawiają się już na poziomie morfologii i składni. Przykładem są idiomy, np. "na wskroś", "na kolanach", "bądź co bądź" oraz powiedzenia takie jak "precz z łapami". Powinny one zostać zinterpretowane przed parsingiem, aby wykluczyć ewentualne błędy ujednoznacznień. Druga klasa obejmuje z kolei wyrażenia semantycznie idiosynkratyczne, czyli posiadające typową składnię, natomiast wyróżniające się swoim znaczeniem. Okazuje się też, że polskie wyrazy wielosegmentowe czasami wymykają się kategoryzacji zaproponowanej przez Saga i in. [86] – problem ten zostanie jeszcze omówiony w niniejszej pracy w rozdziale 6.

W pracy [29] również przedstawiono problemy, które sprawiają polskie MWE, ze szczególnym uwzględnieniem reguł odmiany i zgodności form gramatycznych poszczególnych segmentów dla wyrażeń rzeczownikowych. Autorzy porównują dwa alternatywne formaty opisu polskich WW: Multiflex [88] i POLENG. Pierwszy z nich wspomniano już przy okazji NER – opiera się on na grafowej reprezentacji możliwych wariantów danego WW. Daje to możliwość zapisu wyrazów będących składnikami innych, a także posiadających skróty i szyk przestawny ("większość absolutna" – "absolutna większość"). Format ten jest natomiast dość skomplikowany i operacje na nim wykonywane są ręcznie przy użyciu GUI [47], a ponadto nie pozwala on na opis MWE, w których poszczególne segmenty mogą wystąpić w zdaniu w sposób nieciągły, np. "robić (*go*) w konia", "wolna (*i niczym nie skrępowana*) wola". Format POLENG jest z kolei stosowany w zamkniętym systemie tłumaczącym i jest dużo bardziej zwięzły, natomiast jest on trudniejszy do ręcznej i mechanicznej interpretacji, a ponadto nie pozwala na opis niektórych zjawisk takich jak przestawność szyku. Zapewnia on z kolei możliwość opisu wyrażeń nieciągłych.

Alternatywę dla powyższych formatów, czyli format CB-MWE oparty na języku ograniczeń WCCL (*Wrocław Corpus Constraint Language*) zaproponowano w pracy [38]. Język ten oparty jest na XML i umożliwia definiowanie złożonych warunków, które muszą spełniać poszczególne segmenty – pozwala to na opis różnych zjawisk takich jak przestawność szyku, nieciągłość i inne skomplikowane reguły gramatyczne. Bardzo duża funkcjonalność powoduje niestety rozrost opisu wyrazów – nawet samo zrozumienie reguł dla bardziej skomplikowanych wyrażeń może być kłopotliwe. Opisany format został zastosowany do ręcznego opisania kilku tysięcy haseł polskiego WordNetu [49], który jest jedynym publicznie dostępnym słownikiem dla języka polskiego zawierającym MWE – niestety jego zakres jest mocno ograniczony.

W pracy [109] przedstawiono próbę automatycznego stworzenia słownika polskich WW będącego

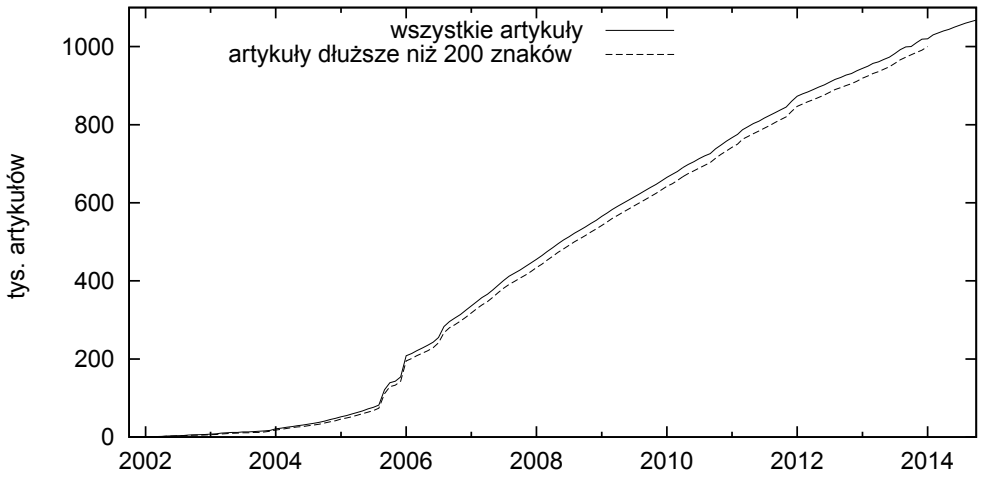
rozwinięciem Słownika Fleksyjnego Języka Polskiego [45, 26]. Zaproponowany format słownika jest bardzo prosty i bazuje na formacie CLP – przykładowo wyraz "ciśnienie atmosferyczne" otrzymuje wzorzec odmienności segmentów "**", która oznacza, że oba segmenty są odmienne oraz etykietę fleksyjną "AE", oznaczającą rzeczownik rodzaju nijakiego. Sam proces ekstrakcji WW opiera się na wykorzystaniu trzech prostych reguł opisujących strukturę syntaktyczną typowych WW: *rzeczownik + przymiotnik*, *przymiotnik + rzeczownik* i *rzeczownik + rzeczownik w dopełniaczu*. Statystyczne miary współwystępowania i reguły heurystyczne posłużyły do wyekstrahowania ponad 10 tys. WW z próbki korpusu IPIPAN [72]. Wartość F_1 samego procesu walidacji kandydatów na WW wyniosła 75%, jednak nie przeprowadzono późniejszych testów rozpoznawania tych WW w tekście.

Analiza powyższych prac prowadzi do wniosku, że dla języka polskiego z pewnością potrzebny jest pewien format opisu WW. Wszystkie powyższe propozycje były rozważane i ostatecznie zdecydowano się na format opisywany w ostatniej pracy, który dodatkowo uproszczono. Poza kwestią prostoty głównym argumentem był fakt, że zakres niniejszej pracy ograniczono do wyrażeń rzeczownikowych (patrz rozdział 6), a ponadto zrezygnowano z opisu bardziej skomplikowanych zjawisk takich jak przestawność szyku. Co więcej, jednym z celów niniejszej pracy jest rozwój Słownika Fleksyjnego Języka Polskiego, w związku z czym kompatybilność formatów poszczególnych narzędzi powinna zostać zachowana.

2.3. Wikipedia jako źródło danych leksykalnych i semantycznych

Podstawowym źródłem danych wykorzystywanych w niniejszej pracy jest polska Wikipedia [105]. Jest to częściowo ustrukturyzowany zasób leksykalny – artykuły posiadają strukturę encyklopedyczną, jednak jest ona luźna i nie zawsze przestrzegana. Przeważająca część artykułów encyklopedycznych o tytułach będących nazwami lub terminami zawiera definicję desygnatu – kilka słów, które w dość dokładny sposób określają, czym jest opisywany obiekt. Zwykle definicja zawiera rzeczownik pospolity, np. artykuł dotyczący Krakowa zawiera na początku słowo "miasto", które zarówno definiuje opisywany obiekt, jak i pozwala na zaliczenie go do klasy miast. Ponadto Wikipedia ma strukturę sieci – artykuły połączone są różnego rodzaju linkami i zaliczane do różnych kategorii. W końcu treść artykułów może posłużyć za korpus języka polskiego. Widzimy więc, że zawartość Wikipedii może posłużyć do ekstrakcji wyrazów w tym wielosegmentowych (hasła), etykiet semantycznych (definicje), relacji semantycznych (przekierowania, linki, kategorie) oraz do trenowania algorytmów statystycznych (treść jako korpus).

Wikipedię spośród polskich encyklopedii wyróżnia całkowicie otwarty dostęp do danych w postaci elektronicznej z możliwością ich dodawania i edycji przez dowolnych użytkowników. Powoduje to, że pod względem ilości artykułów przewyższa ona znacznie inne encyklopedie – obecnie znajduje się w niej ponad milion artykułów. Dla porównania, w 30-tomowej Wielkiej Encyklopedii PWN znajduje się około 140 tysięcy haseł. Tempo wzrostu liczby artykułów zobrazowano na rys. 2.1 – wykres ten może posłużyć do analizy kompletności informacji. Wyraźnie widać, że na przełomie lat 2005 i 2006 nastąpił gwałtowny rozwój, natomiast od roku 2007 zauważamy powolny spadek szybkości dodawania nowych artykułów, co pozwala sądzić, że etap burzliwego rozwoju już minął, a Wikipedia jest dojrzałym i kompletnym



Rysunek 2.1: Wzrost liczby artykułów polskiej Wikipedii[5].

projektem. Ważna jest informacja, że artykuły o długości do 200 znaków, które zwykle są niekompletne, stanowią jedynie niewielką jej część. Dużą zaletą Wikipedii jest szybka reakcja na bieżące wydarzenia – odpowiednie hasła są często modyfikowane już po kilku godzinach, a nawet minutach. Pozwala to na częstą aktualizację danych językowych ekstrahowanych z tej encyklopedii.

Fakt, że zmiany mogą być wykonywane przez przypadkowe osoby, jest jednak przyczyną nierównomiernej jakości artykułów. Okazuje się, że wiele stron jest niekompletnych lub napisanych w niewłaściwym stylu. Częste są błędy ortograficzne i gramatyczne, które mogą uczynić artykuł niemożliwym do automatycznego przetworzenia. Istnieje też ryzyko celowego zafałszowania informacji czy też wandalizmu (powstały jednak mechanizmy zapobiegania tym problemom[6]). Ponadto część artykułów to zestawienia, podsumowania, listy i tabele, które nie opisują konkretnych obiektów. Wszystkie tworzone algorytmy muszą więc działać mimo obecności wyjątków, błędów i niespójności. Mimo to jednak wydaje się, że jakość Wikipedii stale wzrasta, a jej wady nie są na tyle istotne, by uniemożliwiły automatyczną ekstrakcję użytecznych danych.

Różne wersje językowe Wikipedii są coraz częściej używane do ekstrakcji danych leksykalnych, uzupełniając lub wręcz zastępując inne źródła danych. Przykładem mogą być zagadnienia wyznaczania semantycznego powiązania wyrazów oraz ujednoznacznianie słów. WordNet [21] jest typowym zasobem używanym w takich zastosowaniach, jednak jest on krytykowany z powodu niskiego pokrycia, szczególnie nazwami własnymi i nadmiernie skomplikowanej klasyfikacji bytów – definiowane znaczenia są tak drobnoziarniste, że nawet człowiekowi ciężko jest je rozróżnić [54]. Poza tym WordNet poprzez koncepcję *synsetów* (zbiorów synonimów) skupia się na podobieństwach między słowami (jak tezaurus), a nie na różnych znaczeniach tego samego słowa (jak słownik) [18]. Tych wad nie posiada Wikipedia, której hasła są kategoryzowane w przejrzysty sposób, znaczenie każdego hasła opisane jest prosto, a strony

[5] Statystyki Wikipedii: `http://stats.wikimedia.org/EN/TablesWikipediaPL.htm`
oraz: `http://pl.wikipedia.org/wiki/Wikipedia:Liczba artykułów w polskojęzycznej Wikipedii`

[6] Np. system wersji przejrzanych, `http://pl.wikipedia.org/wiki/Wikipedia:Wersje_przejrzane`

ujednoznaczniające wskazują jasno możliwe znaczenia danego słowa. Oczywiście wadą jest brak formalnej definicji całej struktury i przeznaczenie jej do czytania przez człowieka, a nie przez maszyny.

Zastąpienie WordNetu Wikipedią w typowych zagadnieniach przetwarzania języka naturalnego często prowadzi do poprawy wyników. Przykładem może być problem oceny powiązania semantycznego par wyrazów. Typowy test polega na porównaniu korelacji uzyskanych wyników ze zbiorem testowym oznaczonym przez człowieka, np. WS-353 [22] zawierającym 353 pary wyrazów. W pracy [70] przedstawiono wyniki obliczania powiązania semantycznego przy pomocy różnych źródeł danych. Klasycznym źródłem jest WordNet, którego synsety są powiązane ze sobą różnymi relacjami. Alternatywą jest Wikipedia, której sieć kategorii stron również ilustruje powiązania między słowami. Okazuje się, że dla zbioru WS-353 wynik korelacji uzyskany dla Wikipedii to 0.64, a dla WordNetu – 0.38. W obydwu przypadkach użyto kilku algorytmów obliczania powiązania słów i wybrano najlepszy z nich, więc wynik jest miarodajny. Powyższy problem jest dość popularny, więc był rozwiązywany wieloma metodami, między innymi przy użyciu techniki LSA (*Latent Semantic Analysis*) [41]. Posługuje się ona macierzą obrazującą występowanie słów w dokumentach i dokonuje uproszczenia tej macierzy metodą SVD (*Singular Value Decomposition*), po czym wektory dotyczące danych słów mogą zostać porównane przy pomocy miary kosinusowej w celu obliczenia powiązania. Metoda ta na powyższym zbiorze testowym osiągnęła wynik korelacji 0.55 [22]. W pracy [25] przedstawiono alternatywę dla LSA – metodę ESA (*Explicit Semantic Analysis*), która wykorzystuje skonstruowane dla danego wyrazu wektory artykułów Wikipedii z nim powiązanych. Główną zaletą tego podejścia jest wykorzystanie zorganizowanego zasobu wiedzy zamiast zwykłego tekstu, co pozwala na osiągnięcie wysokiej korelacji 0.75.

Kolejnym przykładem udanego użycia Wikipedii jest metoda wikifikacji (*wikify*) [53], polegająca na automatycznym oznaczaniu w tekście linków do haseł Wikipedii, naśladując ręczny proces wstawiania analogicznych linków w artykułach Wikipedii. Algorytm wykorzystuje słownik utworzony z haseł Wikipedii do ekstrakcji kandydatów, a następnie dokonuje ich rankingu stosując miary statystyczne (m.in. χ^2 i TF-IDF). Dodatkowo autorzy wykorzystali rezultaty tej metody do stworzenia skutecznego systemu ujednoznaczniania znaczenia słów (*word sense disambiguation*) – uzyskana wartość F_1 wyniosła 80.7%. Warto odnotować, że również dziedzinie ujednoznaczniania słów stosowano WordNet, jednak wyniki były ograniczone [102], więc Wikipedia ponownie rozszerza znacząco dotychczasowe możliwości. Koncepcja wikifikacji została dalej rozwinięta w pracy [55] – sam etap ujednoznaczniania osiągnął wysoką wartość F_1 wynioszącą 97%.

Wikipedia może też być używana do tworzenia ontologii. Przykładem jest YAGO (*Yet Another Great Ontology*) [94] – ontologia zawierająca 1.7 miliona obiektów i 15 milionów faktów. Powstała ona w oparciu o przetwarzanie infoboksów i kategorii artykułów Wikipedii, dodatkowo wspierając się WordNetem. Infoboksy to ramki z ustrukturyzowaną informacją, tworzone wg ustalonych szablonów. Używane są one do ekstrakcji relacji oznaczających fakty – w tym tak szczegółowe, jak np. posiadania przez dane państwo PKB o danej wartości w określonym roku. Z kolei kategorie pozwalają zorganizować strony w hierarchię typów, przy czym powiązanie strony z kategoriami często jest dość luźne, np. hasło "Elvis Presley" należy m.in. do kategorii "Mississippi Blues Trail" i "Deaths from myocardial infarction". Kategorie stron są

mapowane do synsetów WordNetu, co pozwala umieścić je w zorganizowanej taksonomii.

Innym przykładem bazy wiedzy tworzonej z Wikipedii jest DBPedia [42]. Zawiera ona dane wyekstrahowane z wielu wersji językowych Wikipedii wraz z powiązaniami między nimi. Ekstrahowane są m.in. fakty z infoboksów, współrzędne geograficzne, linki i przekierowania między artykułami, kategorie stron itp. W przypadku języka polskiego warto też wymienić NELexicon [46] – leksykon zawierający ponad 1.4 mln nazw własnych przypisanych do kategorii. Głównym źródłem danych była Wikipedia. Zasób ten powstał niedawno i zawiera m. in. opisy kategorii poszczególnych obiektów, a także ich linki przychodzące w surowej formie. Wydaje się jednak, że DBPedia stanowi obecnie bogatsze źródło danych.

Powyżej wymieniono tylko kilka przykładowych zastosowań Wikipedii w przetwarzaniu języka naturalnego. W rzeczywistości jej zastosowania są dużo szersze – liczne interesujące przykłady można znaleźć w pracy [51]. Wymienione cechy Wikipedii oraz sukcesy w automatycznej ekstrakcji informacji językowej z tej encyklopedii sprawiły, że zdecydowano się na wykorzystanie polskiej Wikipedii zarówno do ekstrakcji etykiet semantycznych, jak też i wyrazów wielosegmentowych.

2.3.1. Wikipedia a NER

Podstawowe zastosowanie Wikipedii do ekstrakcji NE polega na utworzeniu z niej leksykonu. Ponieważ powinien on zostać oznaczony kategoriami NE, potrzebna jest też metoda ekstrakcji tych kategorii. Jednym z możliwych rozwiązań jest analiza treści pierwszego zdania artykułu Wikipedii w celu znalezienia definicji hasła. Można w tym celu dodatkowo posłużyć się WordNetem: dla każdego rzeczownika z pierwszego zdania artykułu próbujemy wyznaczyć kategorię, do której należy przy pomocy WordNetu, a następnie wybieramy dominującą kategorię. Przykładowo dla zdania “Portugal is a country in southwestern Europe” rzeczowniki “country” i “Europe” oznaczają miejsca, więc również hasło “Portugal” oznaczymy tą kategorią. Autorzy pracy [99] posłużyli się tym podejściem do utworzenia leksykonu opartego na Wikipedii w uproszczonym języku angielskim[7]. Rozwinięcie tego podejścia można znaleźć w pracy [33] – tutaj dokonano częściowego parsingu pierwszego zdania artykułu i wybierano pierwsze wyrażenie nominalne występujące po formie słowa “to be”, pobierając bardziej szczegółową definicję.

Powyższe publikacje stały się bezpośrednią inspiracją podczas tworzenia **algorytmu ekstrahującego etykiety semantyczne** dla haseł Wikipedii – dobre wyniki dla języka angielskiego sugerowały, że i dla j. polskiego można uzyskać pozytywne rezultaty, co też zostało osiągnięte. Ponieważ utworzony algorytm został już opisany w pracach [10, 11], w niniejszej pracy opisano go jedynie w skróconej formie z naciskiem na różnice w stosunku do poprzedniej wersji. Algorytmowi temu poświęcony jest rozdział 5.

Wracając do zagadnienia NER, autorzy pracy [33] dodatkowo przetestowali tager oparty na CRF, w którym dodatkowymi cechami stały się kategorie z utworzonego leksykonu. Uzyskana wartość F_1 dla NER na zbiorze CoNLL-2003 dla języka angielskiego to 88% – jest to jeden z wyższych wyników uzyskanych na tym zbiorze. Oczywiście zastosowanie powyższej metody jest możliwe tylko wtedy, gdy istnieje wzorcowy korpus treningowy. W pracy [62] zaproponowano jednak odmienne podejście, polegające na traktowaniu samej treści artykułów Wikipedii jako korpusu treningowego, w którym linki

[7] Simple English Wikipedia, `https://simple.wikipedia.org`

do innych haseł są tagami oznaczającymi wystąpienia NE. Następnie posłużono się tym korpusem do wytrenowania tagera rozpoznającego NE w tekście. W celu podniesienia jakości tworzonego korpusu autorzy wykorzystali dodatkowe reguły heurystyczne. Sukces był częściowy – uzyskane wartości F_1 wahają się w zakresie 60-75% w zależności od metody treningowej i korpusu testowego. Inni autorzy wykorzystują dodatkowo kategorie artykułów, przekierowania i linki *interwiki*[8][83]. Podejście takie pozwala na osiągnięcie wysokiego wyniku F_1 przekraczającego 80% dla kilku różnych języków, jednak wyniki te zostały zweryfikowane w dużo mniejszym stopniu niż w poprzedniej pracy.

W ostatnich latach, gdy postęp niniejszej pracy był już znaczny, zaczęły pojawiać się prace na temat NER przy użyciu Wikipedii dla języka polskiego – przykładem jest podejście opisane w artykule [68]. Opiera się ono na dalszym rozwinięciu koncepcji wikifikacji [55] i dostosowaniu jej do języka polskiego. Uzyskane wyniki testu NER to precyzja 90% i pełność 42%, co skutkuje wartością F_1 wynoszącą 57%. Niska pełność wynika z ograniczenia leksykonu do Wikipedii. Unikalną cechą pracy jest wykorzystanie ontologii CYC [48] celem zwiększenia precyzji kategoryzacji semantycznej używanej przy ekstrakcji NE. Ponieważ zasób ten jest niedostępny dla języka polskiego, wykorzystano linki międzyjęzyczne Wikipedii do odwołania się do j. angielskiego, a następnie do pojęć CYC.

Powyższe prace w dalszym ciągu opisują rozpoznawanie NE należących do wybranych kategorii, więc nie można użyć ich wprost do ekstrakcji WW dla języka polskiego. Niektóre pomysły z całą pewnością mogą jednak zostać wykorzystane.

2.3.2. Wikipedia a ekstrakcja MWE

Dużo rzadsze niż w przypadku NER są prace opisujące ekstrakcję dowolnych MWE przy pomocy Wikipedii – istniejące publikacje ograniczają się zwykle do wyboru pewnej liczby MWE pasujących do z góry ustalonych reguł gramatycznych spośród haseł Wikipedii i zbadania ich dokładności [5]. Interesująca jest praca [3], w której nacisk położony został na rozpoznawanie wśród haseł Wikipedii semantycznie niedekomponowalnych rzeczownikowych MWE. W tym celu wykorzystano zjawisko CCA (*Crossligual Correspondence Asymmetries*), które polega na tym, że wyraz wielosegmentowy (np. "panna młoda") w jednym języku często tłumaczy się na jednosegmentowy ("bride") w innym. Pozwoliło to przy użyciu linków *interwiki* ekstrahować wybraną klasę MWE z dużą precyzją i F_1 przekraczającym 60%. Niestety nie wykonano testu rozpoznawania wynikowych MWE w tekście – ograniczono się wyłącznie do ekstrakcji wyrazów.

W pracy [101] opisano korpus Wiki50, zawierający 50 artykułów angielskiej Wikipedii oznaczonych wystąpieniami MWE i NE. Jest to ważny krok w kierunku utworzenia otagowanych zasobów zawierających MWE, lecz rozmiar tego zbioru jest nieduży, a ponadto dotyczy on tylko języka angielskiego. Autorzy podobnie jak w [62] próbują też ekstrahować rzeczownikowe MWE z artykułów Wikipedii na podstawie linków między artykułami, uzyskując wartość F_1 równą 56% po dodaniu reguł heurystycznych. Dodatkowo podjęto próbę rozpoznawania MWE i NE przy pomocy tagera opartego na CRF wykorzystującego utworzony korpus do treningu. Dla MWE uzyskano wartość F_1 równą 63%.

[8] Linki między Wikipediami w różnych językach.

Podsumowując dotychczasowe wyniki możemy stwierdzić, że nie znaleziono publikacji opisujących rozwiązanie problemu ekstrakcji MWE, gdy nie są dostępne bogate korpusy treningowe, zwłaszcza dla języka polskiego. Uzasadnione jest więc użycie Wikipedii. Wśród wymienionych prac najbliżej rozważanego problemu są artykuły [62] i [101] – użycie linków wydaje się dobrym podejściem do wykrycia wystąpień nie tylko NE, ale także i MWE. Niestety opisane reguły heurystyczne [62] zoptymalizowane dla określonych kategorii NE nie nadają się do ogólnego problemu ekstrakcji MWE. Ponadto dokładność tagera MWE wytrenowanego na linkach byłaby prawdopodobnie dużo niższa niż opisywana w tej pracy dokładność rozpoznawania konkretnych klas NE. Oprócz tego potrzebujemy metody nie tylko rozpoznającej MWE, ale także zwracającej wyekstrahowane informacje syntaktyczne (wzorzec odmiany). Istnieje więc konieczność opracowania nowych algorytmów.

Widzimy więc, że badania nad nowymi metodami realizującymi nakreślone cele są **w pełni uzasadnione**, a niniejsza praca ma w dużej mierze charakter eksploracyjny, ponieważ nie istnieją poprzednie wyniki będące punktem odniesienia. Przeprowadzone badania pozwolą stwierdzić, z jaką dokładnością można ekstrahować przy pomocy Wikipedii wyrazy wielosegmentowe z nieuporządkowanego tekstu polskiego bez użycia otagowanych zbiorów treningowych, ręcznie tworzonych reguł i korzystających z nich klasyfikatorów i tagerów. Badania takie nie były jeszcze prowadzone, a ich efektem powinno być nie tylko wyznaczenie punktu odniesienia (*baseline*) dla dalszych prac, ale również stworzenie brakujących zasobów zawierających MWE dla języka polskiego. Głównym wyzwaniem będzie przygotowanie nowych algorytmów pozwalających na automatyczną ekstrakcję MWE korzystając jedynie z artykułów Wikipedii.

Dokładne przedstawienie rozważanego zagadnienia ekstrakcji WW dla języka polskiego przedstawiono w rozdziale 6, a rozdziały 7-10 prezentują proponowane metody rozwiązania tego problemu i uzyskane wyniki.

3. Narzędzia słownikowe

Przetwarzanie języka naturalnego wymaga użycia zasobów słownikowych. Podstawowym słownikiem wykorzystywanym w niniejszej pracy jest Słownik Fleksyjny Języka Polskiego (SFJP) [45] – decyzję o jego wyborze podjęto jeszcze w roku 2009, gdy praktycznie jedyną alternatywą był analizator Morfeusz [107]. Na podjętą decyzję wpłynął głównie fakt, że w SFJP jednostki słownikowe posiadają swoje unikalne numery ID i jednoznaczne wektory odmiany. Dzięki temu można np. rozstrzygnąć niejednoznaczność między nieodmiennym słowem "pas" oznaczającym rezygnację z ruchu w grze, a słowem odmiennym, oznaczającym podłużny kształt lub przedmiot. Oba wyrazy są rodzaju męskiego nieżywotnego, więc są nierozróżnialne dla Morfeusza. Bazodanowy charakter słownika pozwala ponadto na łatwe dodawanie nowych warstw zawierających np. informacje semantyczne (patrz rys. 1.1 we wstępie). Ponadto jednym z celów pracy jest właśnie rozbudowa narzędzi SFJP i SSJP, więc użycie tego zasobu jest naturalne. Po kilku latach badań okazało się, że słownik ten można wzbogacić, dodając do niego hasła z biblioteki Morfologik [57] i tworząc w efekcie słownik CLPM, opisany w kolejnych podrozdziałach.

Rozważano również użycie tagerów – co prawda wówczas, gdy rozpoczynano badania (rok 2009), podejmowane były jedynie próby tagowania języka polskiego tagerami utworzonymi głównie dla języka angielskiego, dające mieszane wyniki [39], jednak od tego czasu został wykonany duży postęp. Obecnie dostępne są tagery dedykowane dla języka polskiego, m.in. Pantera [2], WMBT [76], WCRFT [74] i Concraft [103], które pozwalają na wzbogacenie informacji leksykalnej poprzez rozpoznawanie słów nieznanych i ujednoznacznianie wyników zwracanych przez słownik. Być może w przyszłości należałoby rozważyć ich użycie, zwłaszcza jeżeli ich poziom poprawności (obecnie nieznacznie przekraczający 90%) znacznie wzrośnie[1].

3.1. Słownik Fleksyjny Języka Polskiego i biblioteka CLP

Słownik Fleksyjny Języka Polskiego posiada interfejs programistyczny – bibliotekę CLP [26] dla języka C. Każdy wyraz w słowniku posiada swój unikalny numer ID. Podstawową operacją udostępnianą przez bibliotekę CLP jest pobranie tego numeru dla podanego słowa, występującego w dowolnej formie fleksyjnej. Realizuje ją funkcja clp_rec. Jako argument przyjmuje ona ciąg znaków, natomiast zwraca tablicę możliwych numerów ID. Przykładowo wywołanie clp_rec("wola") zwraca trzy numery:

[1] Porównanie tych tagerów znajduje się w pracy [69], natomiast próby łączenia tych metod w celu podniesienia skuteczności opisano w pracy [35].

286823840, 286823936 i 286824016, czyli istnieją trzy wyrazy w słowniku, dla których co najmniej jedna z form to "wola".

- 286823840 – rzeczownik rodzaju żeńskiego "wola".
- 286823936 – rzeczownik rodzaju nijakiego "wole".
- 286824016 – przymiotnik "woli" (od: "wół").

Aby sprawdzić, jakie są formy podstawowe tych słów, wywołujemy funkcję clp_bform. Przykładowo: $clp_bform(286823840)$ = "wola", $clp_bform(286823936)$ = "wole", $clp_bform(286824016)$ = "woli".

Dla każdego numeru ID w słowniku występuje etykieta fleksyjna. Jest to ciąg wielkich liter, który określa część mowy oraz sposób odmiany danego wyrazu. Najważniejszą rolę pełni pierwsza litera, czyli część mowy. Dla rzeczownika istotna jest też litera druga, oznaczająca jego rodzaj. Dalsze znaki etykiety są mniej istotne z punktu widzenia niniejszej pracy. Znaczenie poszczególnych liter przedstawiono w tabeli 3.1. Etykietę wyrazu można pobrać przy pomocy funkcji clp_label. Przykładowo, dla wyrazu "wola" $clp_label(286823840)$ = "ADACBAA", dla "wole": $clp_label(286823936)$ = "AEBAC", a dla "woli": $clp_label(286824016)$ = "CBB".

Każdy wyraz posiadający swój numer ID może występować w różnych formach fleksyjnych. Formy te są numerowane kolejnymi liczbami naturalnymi, przy czym numeracja jest oddzielna dla każdej części mowy. Dla rzeczowników liczby od 1 do 7 odpowiadają kolejnym przypadkom liczby pojedynczej, a liczby od 8 do 14 – przypadkom liczby mnogiej. Przymiotniki odmieniają się nie tylko przez przypadki i

Tabela 3.1: Znaczenie początkowych znaków etykiety fleksyjnej w bibliotece CLP.

Przedrostek etykiety	**Część mowy, rodzaj**	
AA	Rzeczownik	męski osobowy
AB		męski żywotny nieosobowy
AC		męski nieżywotny
AD		żeński
AE		nijaki
AF		plurale tantum męskoosobowy
AG		plurale tantum niemęskoosobowy
B	Czasownik	
C	Przymiotnik	
D	Liczebnik	
E	Zaimek	
F	Przysłówek	
G	Wyraz nieodmienny	

liczby, ale także przez rodzaje, dlatego też możliwych numerów form jest znacznie więcej. Liczby od 1 do 28 odpowiadają liczbie pojedynczej (kolejno przypadki dla rodzaju męskiego żywotnego[2], męskiego nieżywotnego, żeńskiego, nijakiego), a numery od 29 do 42 – liczbie mnogiej (rodzaj męskoosobowy i niemęskoosobowy). Podobna sytuacja występuje dla liczebników, jednak mają one 49 form.

Dla danego numeru ID słownik CLP umożliwia pobranie tablicy numerów form odpowiadających pewnemu łańcuchowi znaków w za pomocą funkcji $clp_vec(ID, w)$. Jako przykład przeanalizujmy wynik wywołania funkcji clp_vec dla numerów ID zwróconych przez funkcję clp_rec dla tekstu “wola”.

– Dla rzeczownika “wola” jedyna możliwa forma to mianownik liczby pojedynczej: $clp_vec(286823840,\text{"wola"})$ = [1].
– Dla rzeczownika “wole” możliwe formy to dopełniacz liczby pojedynczej i mianownik, biernik lub wołacz liczby mnogiej: $clp_vec(286823936,\text{"wola"})$ = [2, 8, 11, 14].
– W przypadku przymiotnika “woli” możliwe formy to mianownik lub wołacz liczby pojedynczej rodzaju żeńskiego: $clp_vec(286824016,\text{"wola"})$ = [15, 21].

Kolejna ważna funkcja to clp_formv. Zwraca ona wektor odmiany, czyli tablicę wszystkich form tekstowych odpowiadających kolejnym formom fleksyjnym danego słowa. Przykładowo, dla numeru 286823840 (rzeczownik “wola”) zostanie zwrócona tablica: ["wola", "woli", "woli", "wolę", "wolą", "woli", "wolo", "wole", "woli", "wolom", "wole", "wolami", "wolach", "wole"]. Z kolei funkcja clp_forms zwraca tablicę wszystkich unikalnych form danego słowa: $clp_forms(286823840)$ = ["wola", "woli", "wolę", "wolą", "wolo", "wole", "wolom", "wolami", "wolach"]. Ostatnia ważna funkcja to clp_form. Zwraca ona formę tekstową danego wyrazu w formie fleksyjnej o podanym numerze. Przykładowo, słowo “wola” w narzędniku liczby mnogiej będzie miało postać: $clp_form(286823840, 7 + 5)$ = "wolami".

Biblioteka CLP stale się rozwija – niedawno dodano imiona i nazwiska, a także informacje morfologiczne. Omówione funkcje stanowią więc jedynie część funkcjonalności, lecz część ta jest wystarczająca w niniejszej pracy.

3.2. Ograniczenia słownika SFJP

Podczas prac nad ekstrakcją wyrazów oraz ich etykiet semantycznych z Wikipedii okazało się, że znaczący odsetek błędnych wyników jest spowodowany brakiem pewnych wyrazów w SFJP. Przykładowe słowa, których w tym słowniku nie znajdziemy, to m.in.:

– nazwy geograficzne, np. “Kraków”, “Podhale”, “Toskania”,
– wyrazy pospolite utworzone od nazw, np. “sądecki”, “kielczanin”,
– wyrazy o skomplikowanej morfologii, generowane według pewnych reguł słowotwórczych, np. “nieliniowość”, “niebieskopióry”, “życiodawczyni”, “łebskość”,

[2]Rodzaj męski żywotny to łączne określenie na rodzaje: męski osobowy i męski żywotny nieosobowy.

– niektóre nowo powstałe słowa, np. “singel”, “komputerowy”, “odtwarzacz”.

Głównym celem podczas tworzenia SFJP była jak najwyższa jakość danych słownikowych, a nie jak najobszerniejsza zawartość. W niniejszej pracy obszerność słownika jest jednak wymagana, aby pełność wyników ekstrakcji była jak najwyższa. Już pierwsze rezultaty badań wykazały, że biblioteka CLP powinna zostać uzupełniona o wyrazy z innego słownika, zawierającego brakujące słowa z powyższych kategorii. Istnieją dwa słowniki (analizatory morfologiczne) mogące posłużyć do rozszerzenia SFJP – Morfeusz [107] i Morfologik [57]. Dane z obu zasobów zostały niedawno połączone w jeden słownik PoliMorf [108], a scalone dane zasiliły słownik Morfologik – zawiera on teraz dane z obu źródeł. W pracy wykorzystano więc słownik Morfologik 2.0 PoliMorf zawierający połączone dane. Cechą odróżniającą Morfologik od biblioteki CLP jest całkowicie odmienny format danych, wykorzystujący **znaczniki morfosyntaktyczne**.

3.3. Znaczniki morfosyntaktyczne

Do zapisu cech morfosyntaktycznych dla języka polskiego często używa się formatu, w którym poszczególne segmenty znakowane są listami skrótów nazw poszczególnych kategorii gramatycznych w języku angielskim. Taki format jest używany np. w Narodowym Korpusie Języka Polskiego (NKJP) [71], bibliotece Morfologik [57] i w analizatorze morfologicznym Morfeusz [107]. Przykładowo, po wpisaniu w interfejsie Morfeusza[3] tekstu “wola” otrzymujemy informacje:

Segment	Lemma	Forma
wola	wola	`subst:sg:nom:f`
	wole	`subst:sg:gen:n2`
		`subst:pl:nom.acc.voc:n2`
	woli	`adj:sg:nom.voc:f:pos`

Trzy różne formy podstawowe (*lemma*) odpowiadają w tym przypadku trzem numerom ID zwróconym przez CLP[4]. Znaczniki formy wskazują z kolei na wartości kategorii gramatycznych, zarówno tych ustalonych (np. `subst` – rzeczownik, `f` – żeński), jak i zmiennych (np. `nom` – mianownik, `sg` – liczba pojedyncza). Kategorie mogą być niejednoznaczne – wówczas jeżeli to możliwe, stosuje się zapis z kropkami, np. zapis `nom.acc.voc` oznacza mianownik, biernik lub wołacz. Ponieważ ten format nie pozwala na zapis dowolnych kombinacji różnych kategorii, w bardziej skomplikowanych przypadkach trzeba podawać kilka alternatywnych znaczników (jak w powyższym przykładzie dla formy podstawowej “wole”).

[3] Wersję online można znaleźć pod adresem `http://sgjp.pl/demo/morfeusz`

[4] Wyrazy z różnymi numerami ID w CLP mogą też posiadać tę samą formę podstawową, np. “pilot” to osoba, ryba lub urządzenie – każdy z tych wyrazów ma inny numer ID, ponieważ różnią się etykietami fleksyjnymi.

3.4. Słownik CLPM

Połączenie słowników wymagało znalezienia metody na zapis danych jednocześnie z obu zasobów. Format słownika CLP ma istotną zaletę – każda jednostka słownikowa identyfikowana jest jednoznacznie przez numer ID, co pozwala np. na oddzielną reprezentację słów "rząd" (dop. "rządu") i "rząd" (dop. "rzędu"). Problemem są z kolei oboczności, które, ponieważ wektor odmiany jest jednowymiarowy, muszą być reprezentowane również przez osobne numery ID, np. słowo "dzieciarnia" ma dwie formy dopełniacza liczby mnogiej: "dzieciarni" i "dzieciarń", więc będzie posiadało dwa różne numery ID. Z kolei w Morfologiku nie występuje pojęcie jednostki słownikowej – tutaj dla każdego słowa dana jest jedynie forma podstawowa (*lemma*) w postaci tekstowej. Uniemożliwia to odseparowanie słów posiadających taką samą formę podstawową i wartości ustalonych kategorii gramatycznych (np. przytoczony przykład "rząd"). W związku z tym postanowiono utworzyć nowy format, w którym również dane ze słownika CLP zostaną scalone tak, aby zachować spójność między tymi dwoma źródłami danych. Testy wykazały, że nie prowadzi to do żadnego obniżenia skuteczności wykorzystywanych w niniejszej pracy algorytmów.

W momencie tworzenia tego nowego słownika prace nad algorytmami ekstrahującymi wyrazy wielosegmentowe były już tak zaawansowane, że ważne stało się zachowanie kompatybilności wstecznej ze słownikiem CLP – osiągnięto ją stosując zbliżony interfejs. Numery ID zostały zastąpione identyfikatorami składającymi się z formy podstawowej, myślnika i początkowego fragmentu etykiety fleksyjnej, natomiast koncepcja wektora form została rozszerzona o oboczności – na każdej pozycji zamiast pojedynczej formy tekstowej może wystąpić ich lista. W porównaniu z Morfologikiem zaletą tego formatu jest zwięzłość, dużo wyższa niż w przypadku znaczników morfosyntaktycznych, natomiast wadą – nieco mniejsza czytelność wynikająca z użycia numerów form. Z kolei w porównaniu z CLP zaletą jest większa czytelność (brak sztucznych numerów ID) i łatwiejszy zapis oboczności, natomiast wadą – wspomniane już scalanie niektórych wyrazów (przykład słowa "rząd"). Scalanie to występuje jednak niezwykle rzadko i w przyszłości można by dodać do słownika odpowiednią listę wyjątków.

Ponieważ niniejsza praca nie wymaga przetwarzania wszystkich części mowy z Morfologika, wybrano tylko rzeczowniki, przymiotniki, imiesłowy czynne i bierne, a także przyimki[5]. Poniżej podsumowano zawartość i format powstałego słownika **CLPM** z podziałem na części mowy:

Rzeczowniki pobierane są zarówno z CLP, jak i z Morfologika. W CLP pierwsza litera etykiety to A, natomiast druga litera oznacza rodzaj (patrz tab. 3.1). Z kolei w Morfologiku stosowany jest znacznik postaci `subst:`**`liczba`**`:`**`przypadek`**`:`**`rodzaj`**[6]. Rodzaj jest ustalony dla danego rzeczownika, natomiast liczba i przypadek ulegają zmianie – wartości stałe tłumaczone są na etykietę, a zmienne na numer formy. W słowniku CLPM pierwsze 2 litery etykiety mają znaczenie takie samo jak w CLP, a dodatkowo okazało się, że istnieje potrzeba wyróżnienia słów odmiennych poprzez przyznanie im odrębnych etykiet. Jako przykład rozważmy rzeczownik męski nieosobowy "pas". Może on

[5] W Morfologiku przyimki zawierają informację o tym, jaki przypadek rzeczownika powinien po nich wystąpić, co jest wykorzystywane w algorytmie tworzącym wzorce syntaktyczne w metodzie SM (rozdział 9).

[6] Pogrubioną czcionką oznaczono nazwy kategorii, pod które trzeba podstawić ich wartości, a zwykłym krojem – niezmienne ciągi znaków.

oznaczać podłużny przedmiot (wyraz odmienny) lub rezygnację z dalszej gry (wyraz nieodmienny). Gdyby wziąć pod uwagę jedynie dwie pierwsze litery etykiety, to oba wyrazy miałyby identyfikator `AC-pas`, co spowodowałoby uznanie form słowa nieodmiennego za oboczności słowa odmiennego. Testy pokazały, że prowadzi to do wielu błędów, zdecydowano się więc na dodanie trzeciej litery do etykiety: litera A oznacza rzeczownik odmienny, a B – nieodmienny. W CLP stosowany jest podobny schemat (np. dla rodzaju męskiego osobowego przedrostek AAI oznacza słowo nieodmienne), więc dodanie tej litery dla słów pobieranych z CLP nie jest trudne. Gorzej jest w przypadku Morfologika – tutaj forma podstawowa "pas" posiada zbiór form wymieszanych dla słowa odmiennego i nieodmiennego. Zaimplementowano więc algorytm, który rozpoznaje takie sytuacje (identyczna forma dla wszystkich przypadków) i rozdziela tego rodzaju rzeczowniki. W wynikowym słowniku znajdą się więc identyfikatory: `ACA-pas` (odmienny) i `ACB-pas` (nieodmienny).

Dodatkowo zdecydowano się pobrać z Morfologika rzeczowniki odczasownikowe (odsłowniki). Posiadają one znacznik zawierający więcej kategorii niż w przypadku zwykłych rzeczowników: `ger:`**`liczba`**`:`**`przypadek`**`:`**`rodzaj`**`:`**`aspekt`**`:`**`zanegowanie`**`:`**`zwrotność`**. Ważną różnicą w stosunku do rzeczowników jest to, że *lemma* jest czasownikiem, więc po pierwsze należy za formę podstawową przyjąć mianownik liczby pojedynczej, a po drugie – od jednego czasownika mogą pochodzić dwa rzeczowniki, w tym jeden zanegowany, np. "palenie" i "niepalenie". W takich przypadkach należy odpowiednio rozdzielić formy w zależności od wartości kategorii *zanegowanie*.

Dla rzeczowników wykonano jeszcze dodatkową operację, polegającą na usunięciu niektórych krótkich rzeczowników nieodmiennych (np. "do", "el", "in"), ponieważ zaburzały one wyniki działania algorytmów. Usunięto również nazwiska pobierane z CLP, które są rzeczownikami pospolitymi, np. "Droga". Pozwoliło to zmniejszyć złożoność ekstrakcji WW poprzez redukcję niejednoznaczności, a przy tym nie obniżyło jakości wyników.

Czasowniki pobierane są tylko z CLP, ponieważ nie pełnią one istotnej roli w niniejszej pracy. Etykieta w CLPM to B, czyli pierwsza litera etykiety z CLP. Z wektora form usuwany jest imiesłów czynny i bierny, ponieważ obecność tych form powoduje problemy w niektórych algorytmach.

Przymiotniki pobierane są również z obu źródeł. W przypadku CLP posiadają one etykiety zaczynające się od CC (odmienne) lub CD (nieodmienne). Te prefiksy uznano za etykiety CLPM. W Morfologiku z kolei występuje format znacznika `adj:`**`liczba`**`:`**`przypadek`**`:`**`rodzaj`**`:`**`stopień`**. Pierwsze trzy kategorie służą do wyznaczenia numerów form, a stopień jest pomijany.

Imiesłowy czynne i bierne ekstrahowane są z obu słowników. W CLP mają one prefiks CB (czynny) i CA (bierny). Prefiksy te uznano za etykiety CLPM. Z kolei w Morfologiku posiadają one znacznik zaczynający się od `pact` (czynny) lub `ppas` (bierny). Kategorie gramatyczne są podobne jak dla rzeczownika odczasownikowego – tutaj również *lemma* jest czasownikiem i występuje problem zanegowania, który rozwiązano analogicznie.

Przyimki w CLP współdzielą z innymi słowami nieodmiennymi przedrostek etykiety G i takie oznaczenie użyto jako etykietę w CLPM. W przypadku Morfologika występuje znacznik `prep:`**`rodzaj`**, w którym znajduje się oczekiwany rodzaj rzeczownika. Informacja ta pozwala na rozpoznawanie wyrażeń przyimkowych i zapisywana jest w specjalnym słowniku metadanych towarzyszących CLPM (nie jest częścią etykiety). Wektor form jest oczywiście jednoelementowy.

Pozostałe kategorie : liczebniki, zaimki, przysłówki i inne wyrazy nieodmienne pobierane są z CLP i oznaczane są etykietą będącą pierwszą literą etykiety z CLP.

Wyekstrahowane wg powyższych kryteriów trójki $(forma_podst, etykieta, wektor_form)$ są zapisywane do bazy danych CLPM. Gdy to samo słowo występuje w CLP i Morfologiku, wektory form są scalane. Jeżeli w wektorze występują oboczności, kolejne formy zapisywane są z numerami zwiększanymi o kolejne wielokrotności liczby 64. Przykładowo, wyraz o identyfikatorze `AAA-wierszokleta` posiada trzy możliwe formy mianownika liczby mnogiej: “wierszoklety”, “wierszokleci”, “wierszokletowie”. Zapisywane są one pod numerami 8, 72, 136. Dane są weryfikowane, co pozwala odrzucić np. rzeczowniki nie posiadające formy mianownika (w obu słownikach występują różnego rodzaju drobne błędy).

CLPM udostępnia operacje: $label$, $bform$, rec, $forms$, $formd$, $form$, vec i tag. Działają one analogicznie jak ich odpowiedniki z CLP z poniższymi wyjątkami:

- Funkcja $formd$ zastąpiła $formv$ (wektor form). Zwraca ona słownik[7], którego kluczami są identyfikatory, a wartościami – listy numerów form. Zmiana ta jest podyktowana obocznościami.
- Nowa funkcja tag zwraca słownik, którego kluczami są identyfikatory, a wartościami – listy numerów form. Funkcja ta łączy więc w sobie rec i vec, pozwalając dzięki temu zminimalizować liczbę operacji odczytu z bazy danych.
- Ponadto funkcje $label$ i $bform$ są bardzo wydajne, ponieważ informacje o etykiecie fleksyjnej i formie podstawowej są zawarte są w identyfikatorze.
- Funkcje rozpoznające formy tekstowe, np. tag i rec biorą pod uwagę wielkość liter, jednak dla argumentu pisanego wielką literą sprawdzany jest dodatkowo wariant pisany małą literą – jest to odzwierciedleniem reguły ortograficznej pozwalającej zapisywać słowa pospolite wielką literą w nazwach i na początku zdań.

Do zapisu zawartości słownika użyto bazy danych LMDB[8], która wyróżnia się bardzo dużą szybkością odczytu. Używane są dwie tabele klucz-wartość. Tabela `Rec` odwzorowuje formę tekstową na zbiór par $(identyfikator, numery_form)$ i pozwala wykonać m.in. operację rec i tag, a tabela `Form` przyporządkowuje identyfikatorowi zbiór par $(forma_tekstowa, numery_form)$, co pozwala zrealizować m.in. operację $formd$.

[7] Ponieważ niniejsza praca implementowana jest w języku Python, termin “słownik” oznacza typ *dict* występujący w tym języku. Alternatywnie możemy uważać słownik za zbiór par klucz-wartość.

[8] Lightning Memory-Mapped Database [43]

Słownik CLPM zawiera 414250 haseł, w tym 406306 unikalnych form podstawowych. Jest to około trzy razy więcej, niż w samym słowniku CLP. Okazuje się, że w np. przypadku rzeczowników i przymiotników niewiele jest słów obecnych w CLP, których nie byłoby w Morfologiku – jest ich zaledwie kilka tysięcy, w tym głównie imiona i nazwiska. Są to jednak wartościowe dane i nie rozważano możliwości całkowitej rezygnacji z CLP. W przyszłości CLPM powinien zostać połączony ze słownikiem WW opracowanym w niniejszej pracy, tworząc nowy słownik fleksyjny dla języka polskiego.

3.4.1. Znaczniki słownikowe

Opisane powyżej znaczniki morfosyntaktyczne mają ważną własność – nadają się dobrze do opisu wyniku rozpoznania formy tekstowej, czyli wykonania operacji tag. Znaczniki te nie pozwalają jednak na jednoznaczną identyfikację hasła w CLPM, a poza tym są nieefektywne pamięciowo i obliczeniowo. Dlatego też dla utworzonego słownika zdefiniowano nowy rodzaj znacznika. W formacie tym uwzględniamy jedynie identyfikatory i wartości zmiennych kategorii gramatycznych, ponieważ kategorie ustalone dla danego wyrazu są funkcją numeru ID – byłyby więc redundantne. Znacznik jest zbiorem par (id, F), gdzie id jest identyfikatorem, a F jest zbiorem numerów form odpowiadającym id. Format ten jest więc równoważny wynikowi zwracanemu przez operację tag. Taki rodzaj znacznika będzie nazywany **znacznikiem słownikowym**. Przykładowo, znacznik słownikowy $tag(\text{"wola"})$ ma postać:

$$\{(\texttt{ADA-wola}, \{1\}), (\texttt{AEA-wole}, \{2, 8, 11, 14\}), (\texttt{CC-woli}, \{15, 21\})\}$$

Poza powyższym formatem będą jeszcze stosowane dwa inne sposoby zapisu takiego znacznika:

- Zbiór par (id, f), gdzie id jest identyfikatorem, a f – pojedynczym numerem formy. Zapis ten polega więc na rozbiciu zbiorów F na pojedyncze elementy. Czasami w niniejszej pracy będziemy dla uproszczenia zakładali, że funkcja tag zwraca zbiór takich par.
- Zapis tekstowy, stosowany w bazie danych. W tym formacie stosuje się pionową kreskę "|" do oddzielenia poszczególnych par (id, F), dwukropek ":" do oddzielenia id i F oraz przecinek "," do oddzielenia elementów zbiorów F. Zapis przykładowego wzorca wygląda następująco:

```
ADA-wola:1|AEA-wole:2,8,11,14|CC-woli:15,21
```

Czasami zachodzi potrzeba utworzenia struktury podobnej do znacznika słownikowego, jednak nie zawierającej identyfikatorów. Jest tak wówczas, gdy np. poszukujemy segmentu posiadającego pewne cechy. Potrzebujemy wówczas znacznika opisującego te cechy. W tym celu wystarczy zastąpić identyfikatory w znaczniku słownikowym przez przedrostki etykiet (patrz tabela 3.1). Otrzymamy wówczas zbiór par (l, F), gdzie l jest przedrostkiem etykiety, a F – zbiorem numerów form. Przykładowo, dla powyższego znacznika słownikowego powstanie zbiór $\{(AD, \{1\}), (AE, \{2, 8, 11, 14\}), (C, \{15, 21\})\}$. Ta struktura nazywana jest **specyfikacją formy**.

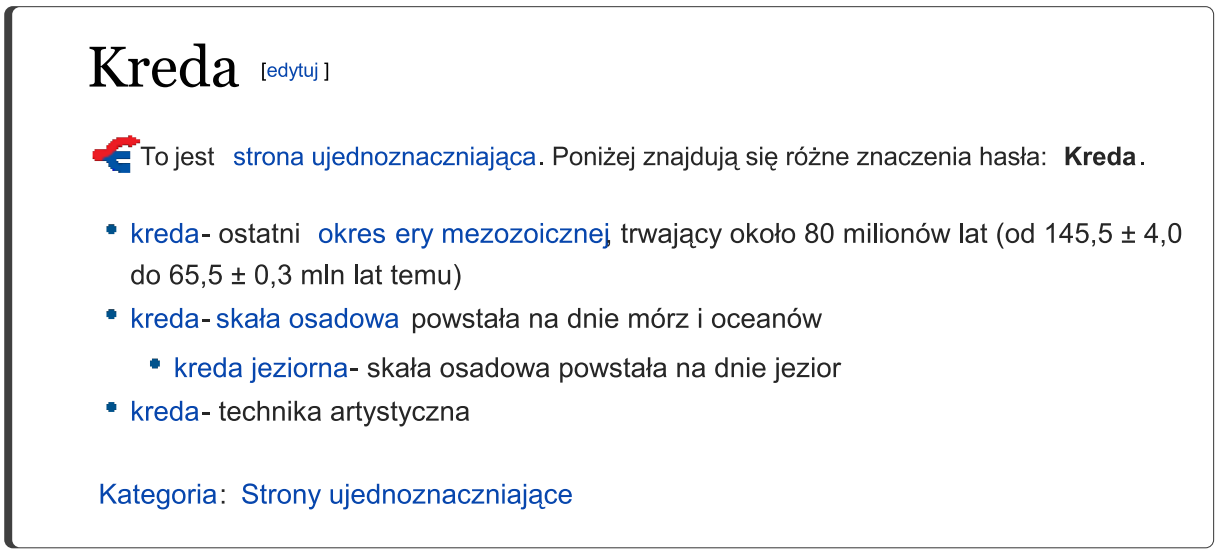
4. Ekstrakcja danych z Wikipedii i Wikisłownika

Polska Wikipedia jest dostępna pod adresem `http://pl.wikipedia.org`. Istnieje więc możliwość pobrania jej stron w sposób automatyczny przy pomocy crawlera. W ten sposób wszystkie strony pobrane zostają w formacie HTML. Rozwiązanie to zostało użyte w pracy magisterskiej Autora [10]. Okazuje się jednak, że obecnie istnieje lepsza metoda: specjalnie na potrzeby automatycznego przetwarzania informacji dostępne są aktualizowane co kilka tygodni zrzuty bazy danych projektów fundacji Wikimedia[1]. Analizując dostępne zasoby zdecydowano, że oprócz Wikipedii wartościowe dane zawiera Wikisłownik [106] – wielojęzyczny słownik, który zawiera informacje na temat odmiany słów. W porównaniu z Wikipedią zawiera on dużo mniej haseł (około 66 tysięcy), uzupełnia jednak dane o wyrazy pospolite i związki frazeologiczne, których często brak w Wikipedii. Mimo że obecnie zasoby Wikisłownika nie są jeszcze wykorzystywane, podjęto decyzję o ekstrakcji ich razem z danymi Wikipedii, a następnie porównaniu użyteczności tych zasobów.

Podobnie jak cały projekt, konstruowanie bazy danych z zasobów fundacji Wikimedia zostało zaimplementowane w języku Python [73]. Pierwszym etapem jest pobranie najnowszej wersji zrzutów Wikipedii i Wikisłownika. Program sprawdza, jaka jest najnowsza pobrana wersja i pobiera nowszą, jeżeli jest dostępna. Na stronach Wikimedia dostępne są różne wersje zrzutów. Dla Wikipedii wybrano wersję `plwiki-pages-articles.xml`. Zawiera ona treść wszystkich artykułów i szablonów. Strony sformatowane są w formacie MediaWiki [52], a wszystkie dane znajdują się w jednym pliku XML. Zdecydowano się na wybór tego pliku, ponieważ zawiera on wszystkie potrzebne dane i w przeciwieństwie do udostępnianych plików SQL nie wymaga bazy MySQL. Ponadto ten format jest wyróżniony na stronie zrzutów jako zalecany – nie zawiera on niepotrzebnych danych takich jak np. historia edycji stron czy też dyskusje wikipedystów, co pozwala oszczędzić miejsce na dysku (plik ma rozmiar 1.2 GB). Analogicznie następuje pobranie danych Wikisłownika – w tym wypadku wybrano wersję `plwiktionary-pages-articles.xml` (rozmiar: 73 MB).

Przetwarzanie danych zrzutu odbywa się przy użyciu biblioteki *mwlib* [59]. Wyekstrahowane dane umieszczane są w relacyjnej bazie danych SQLite [93]. O wyborze tego silnika decydowała szybkość operacji wstawiania i pobierania dużych ilości danych, a także prostota działania i łatwy eksport/import danych (całość znajduje się w jednym pliku). Parser formatu MediaWiki generuje kolejne strony Wikipedii/Wikisłownika, które następnie są przetwarzane i umieszczane w bazie. Strony należą do różnych przestrzeni nazw (*namespaces*), oznaczanych liczbami od 0 do 15. Przestrzeń główna (*main*, 0) oznacza

[1] `http://dumps.wikimedia.org`



Rysunek 4.1: Przykład strony ujednoznaczniającej.

artykuły. Wśród pozostałych przestrzeni warto wymienić strony **szablonów** (*template*, 10) i **kategorii** (*category*, 14).

– **Szablony** zawierają gotowe fragmenty stron, które można wstawić do artykułu, opcjonalnie modyfikując wartości pewnych parametrów. Mogą to być infoboksy (ramki informacyjne), listy stron o podobnej tematyce lub specjalne oznaczenia artykułów, np. artykuły wymagające dopracowania, artykuły łamiące zasadę neutralności itp. Przykładowo, w artykule na temat miasta Kraków znajdziemy m.in. szablony “Miasta województwa małopolskiego”, “Europejskie Stolice Kultury”, “POL miasto infobox” i “Flaga”. Dwa pierwsze służą wyświetleniu listy miast, do których należy Kraków, a dwa kolejne pozwalają wyświetlić podstawowe informacje o mieście i jego flagę.

– **Kategorie** służą z kolei jedynie grupowaniu stron o zbliżonej tematyce, np. artykuł o Krakowie należy m.in. do kategorii “Miasta na prawach powiatu”, “Miejsca obrad polskich sejmów” i “Podział administracyjny województwa małopolskiego”.

W przestrzeni głównej oprócz artykułów znajdują się też strony o specjalnym znaczeniu:

– **Strony ujednoznaczniające** wymagane są wówczas, gdy hasło ma wiele znaczeń. Wówczas istnieje konieczność wprowadzenia strony, która informuje o wieloznacznym haśle i udostępnia linki do poszczególnych pojęć. Przykładem może być strona hasła “Kreda” przedstawiona na rys. 4.1. Ponieważ docelowe artykuły posiadają identyczne nazwy, wprowadzono frazę ujednoznaczniającą zapisaną w nawiasie – artykuły nazywają się więc “Kreda (okres)”, “Kreda (skała)” i “Kreda (technika artystyczna)”.

  Istnieje jeszcze jeden sposób organizacji stron ujednoznaczniających, wykorzystywany wówczas, gdy hasło posiada jedno główne znaczenie. Można wtedy od razu zaprezentować ten najważniejszy artykuł. Przykładem może być słynna z bitwy wieś Racławice. Pod tym hasłem znajduje się zwykły

artykuł, jednak w jego górnej części znajduje się łącze do strony ujednoznaczniającej "Racławice (ujednoznacznienie)", która zawiera listę innych wsi o takiej nazwie – odpowiednie artykuły są zatytułowane "Racławice (powiat krakowski)", "Racławice (powiat gorlicki)" itd.

– **Przekierowania** występują wtedy, gdy różne hasła mają to samo (lub bardzo zbliżone) znaczenie. Przykładem może być hasło "Pies", które zawiera przekierowanie do artykułu zatytułowanego "Pies domowy". Do każdego artykułu może prowadzić wiele przekierowań – hasło docelowe jest zwykle najpopularniejszym (lub najbardziej poprawnym) określeniem na dany obiekt, chociaż nie jest to regułą.

4.1. Proces ekstrakcji danych z Wikipedii

Ekstrakcja rozpoczyna się od przetworzenia szablonów i kategorii, czyli przestrzeni nazw 10 i 14. Są one umieszczane w bazie danych jako pierwsze, aby umożliwić późniejsze dodawanie linków do nich podczas przetwarzania artykułów. Ponieważ ich rola jest inna niż w przypadku stron z artykułami, umieszczane są one w osobnej tabeli `SpecialPage`. Zapisywane atrybuty to nazwa strony (*name*), jej zawartość (*text*) oraz przestrzeń nazw (*namespace*). Wśród szablonów i kategorii mogą wystąpić przekierowania – zapisywane są one w tabeli `SpecialRedir`.

Kolejnym etapem jest przetwarzanie stron z artykułami, czyli przestrzeni nazw 0. Dla każdej strony pobierany jest tytuł (*title*). Następnie przy pomocy wyrażeń regularnych dokonywany jest podział tytułu na nazwę (*name*) i opcjonalną frazę ujednoznaczniającą (*disamb*) – przykładowo dla tytułu "Kreda (okres)" atrybut *name* ma wartość "Kreda", a *disamb* – "okres". Treść strony w formacie MediaWiki przetwarzana jest parserem `DumpParser` z biblioteki *mwlib*, który konstruuje strukturę drzewiastą przypominającą drzewo DOM tworzone przy parsowaniu stron HTML. Powstała struktura wykorzystywana jest do pobrania następujących danych:

– Odwołania do szablonów: przykładowo, do opisu wzoru chemicznego możemy wykorzystać szablon "Chem", np. wzór `{{chem|H|2|SO|4}}` zostanie wyświetlony na stronie jako H_2SO_4. Szablon "Ujednoznacznienie" ma specjalną własność – strony zawierające link do niego są stronami ujednoznaczniającymi.

– Linki do kategorii, np. zapis `[[Kategoria:Kwasy nieorganiczne|Siarkowy, kwas]]` oznacza, że bieżąca strona należy do kategorii "Kwasy nieorganiczne". Parametr "Siarkowy, kwas" oznacza hasło, pod jakim bieżący artykuł wystąpi w automatycznie tworzonym indeksie stron należących do danej kategorii[2].

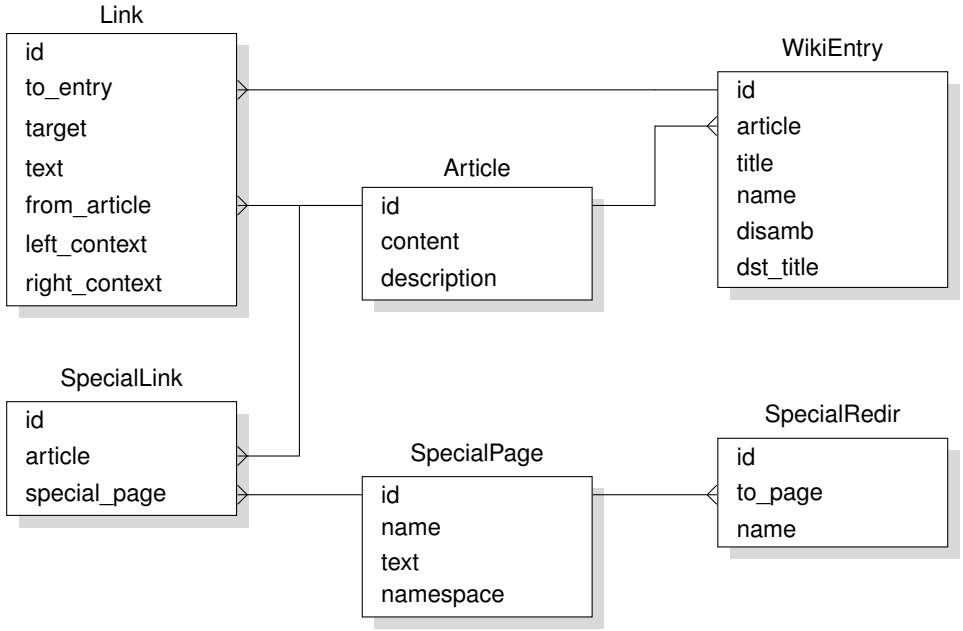
– Linki do innych artykułów – służą jako odnośniki do haseł występujących w tekście, np. zapis "`jeden z najmocniejszych [[Kwasy|kwasów]]`" powoduje, że na stronie znajdzie się wyrażenie

[2]Hasło to służy jedynie do sortowania artykułów, w indeksie kategorii wyświetlany jest po prostu tytuł strony.

"jeden z najmocniejszych kwasów", w którym słowo "kwasów" będzie prowadziło do strony zatytułowanej "kwasy"[3].

– Treść artykułu – pobierana tylko dla stron, które nie są stronami ujednoznaczniającymi. Zawiera ona tekst artykułu, w którym każdy akapit zapisany jest zwykłym tekstem w jednej linii. Pomijane są tabele, podpisy rysunków, przypisy, akapity wcięte lub zapisane czcionką o stałej szerokości oraz listy, a także treści wygenerowane z szablonów.

– Opis hasła – tylko dla stron, które nie są stronami ujednoznaczniającymi. Jest to pierwszy akapit artykułu. Służy on do pobrania definicji hasła. Zachowane jest częściowe formatowanie, np. podwójny apostrof oznacza kursywę, a potrójny – pogrubienie. Ma to na celu ułatwienie znalezienia definicji.



Rysunek 4.2: Struktura bazy danych wyekstrahowanych z Wikipedii.

Jeżeli dana strona jest przekierowaniem, przetwarzanie treści nie ma oczywiście sensu. W takim przypadku największe znaczenie ma tytuł strony docelowej (*dst_title*), który jest zapisywany w bazie danych. Po zakończeniu przetwarzania stron następuje połączenie linków i przekierowań (dopiero wtedy mamy pewność, że wszystkie strony zostały pobrane). Należy wówczas sprawdzić, czy nie ma przekierowań cyklicznych, rozwiązać przekierowania pośrednie oraz usunąć niepoprawne. Zdecydowano, że zamiast przechowywać przekierowania i strony z artykułami osobno, obydwa rodzaje stron zapisywane są w tabeli `WikiEntry`[4]. Zapisywane atrybuty to: tytuł (*title*), nazwa (*name*), fraza ujednoznaczniająca (*disamb*) oraz (dla przekierowań) tytuł strony docelowej (*dst_title*). Same artykuły zapisywane są w osobnej tabeli

[3] W rzeczywistości strona ta nazywa się "Kwasy", ponieważ tytuły pisane są wielką literą.

[4] Ma to na celu uproszczenie wykorzystywanych później zapytań do bazy danych, które musiałyby wykonywać dodatkowe rozwiązywanie przekierowań. Jest to jedna z kilku optymalizacji, które wynikają ze specyfiki algorytmów używanych w kolejnych fazach przetwarzania danych.

`Article`, przechowującej treść artykułu (*content*) oraz opis hasła (*description*). Zamiast powiązań między przekierowaniami a artykułami docelowymi występuje więc relacja jeden-do-wielu między artykułem a stronami. Linki do szablonów i kategorii zapisywane są w osobnej tabeli `SpecialLink` jako powiązania wiele-do-wielu między artykułem (*article*) a stroną specjalną (*special_page*). Analogicznie, tabela `Link` przechowuje linki prowadzące od źródłowego artykułu (*from_article*) do docelowego hasła (*to_entry*). Wraz z linkami przechowywany jest ich cel, czyli tytuł strony docelowej w oryginalnej pisowni (*target*), wyświetlany tekst (*text*) a także ich kontekst (tekst znajdujący się na lewo i prawo od artykułu, w chwili obecnej ograniczony do 64 znaków z każdej strony – *left_context*, *right_context*). Schemat powstałej bazy danych przedstawiono na rys. 4.2.

znaczenia:

rzeczownik, rodzaj męskozwierzęcy

(1.1) *zool.* *Felis domestica*[1], zwierzę domowe; *zob.* też kot domowy w Wikipedii
(1.2) *zool.* każde zwierzę drapieżne z rodziny *Felidae*[2], o smukłym ciele, miękkiej sierści i długim ogonie
(1.3) *łow.* samica zająca
(1.4) *pot.* kłąb kurzu zbierającego się w zakamarkach mieszkania

rzeczownik, rodzaj męskoosobowy

(2.1) *slang.* rekrut, nowy uczeń

odmiana :

(1.1–4) [ukryj▲]

przypadek	liczba pojedyncza	liczba mnoga
mianownik	**kot**	**koty**
dopełniacz	kota	kotów
celownik	kotu	kotom
biernik	kota	koty
narzędnik	kotem	kotami
miejscownik	kocie	kotach
wołacz	kocie	koty

(2.1) [ukryj▲]

przypadek	liczba pojedyncza	liczba mnoga
mianownik	**kot**	**koty**
dopełniacz	kota	kotów
celownik	kotu	kotom
biernik	kota	kotów
narzędnik	kotem	kotami
miejscownik	kocie	kotach
wołacz	kocie	koty

Rysunek 4.3: Sekcje “znaczenia” i “odmiana” dla hasła “Kot” w Wikisłowniku. Znaczenia 1.1 – 1.4 mają taki sam rodzaj i wzorzec odmiany, natomiast znaczenie 2.1 posiada inny rodzaj i wzorzec odmiany.

4.2. Proces ekstrakcji danych z Wikisłownika

Podobnie jak w przypadku Wikipedii, pierwszym krokiem jest pobranie wybranych stron specjalnych (szablonów i kategorii) i zapisanie ich w tabeli `SpecialPage`. Następnie przetwarzane są strony z artykułami. Dla każdej strony pobierany jest jej tytuł (*title*) oraz wybrane cechy wyekstrahowane z treści. Podobnie jak dla Wikipedii, treść przetwarzana jest przy pomocy parsera `DumpParser` do postaci drzewiastej, jednak dalsze przetwarzanie danych przebiega już inaczej. Artykuły w Wikisłowniku są dzielone na sekcje według języka, np. słowo "tak" będzie miało sekcję dla języka polskiego, czeskiego, szwedzkiego itd. Ekstrahowana jest tylko sekcja polska – jeżeli artykuł jej nie posiada, jest pomijany. Treść każdej sekcji jest w pełni ustrukturyzowana – występują podsekcje takie jak "wymowa", "znaczenia", "odmiana" itp. Każda z tych podsekcji jest generowana przy użyciu odpowiedniego szablonu, co pozwala na wyekstrahowanie ustrukturyzowanej informacji z odwołań do szablonów. Kluczowa jest sekcja "znaczenia", ponieważ wprowadza ona identyfikatory poszczególnych znaczeń, które są później używane w kolejnych podsekcjach. Identyfikator znaczenia składa się z liczb oddzielonych kropkami. Przykład dla słowa "Kot" pokazano na rys. 4.3. Występują tu dwie odrębne jednostki leksykalne: "kot" rodzaju zwierzęcego (1.1-4) i osobowego (2.1). Struktura identyfikatorów może też być bardziej skomplikowana, więc zdecydowano, że każde ze znaczeń jest ekstrahowane jako osobny obiekt i zapisywane do tabeli `WiktEntry`.

Zaimplementowany algorytm pozwala jedynie na pobieranie danych dla rzeczowników – jest to jedynie wersja próbna, zaimplementowana w celu zbadania sensu dalszych prac nad automatycznym przetwarzaniem Wikisłownika. Poza ekstrakcją cech rzeczowników i prostą analizą rezultatów nie były wykonywane dalsze prace. Ekstrahowane cechy są przedstawione w tabeli 4.1.

Tabela 4.1: Cechy ekstrahowane z Wikisłownika.

Nazwa	**Opis**	**Przykład**	**Źródło**
name	Słowo	*kot*	tytuł sekcji polskiej
category	Część mowy i kategorie gramatyczne	*rzeczownik, rodzaj męsko-zwierzęcy*	podsekcja "znaczenia"
description	Opis znaczenia	*zwierzę domowe*	podsekcja "znaczenia"
infl_type	Ogólny typ odmiany	*noun* (oznacza deklinację rzeczownika)	podsekcja "odmiana"
forms	Wektor form	*kot, kota, kotu, (...)*	podsekcja "odmiana"
inflection	Opis odmiany, gdy brak wektora form	Dla słowa *zapisać* jest odnośnik *zob.: zapisywać*	podsekcja "odmiana" zapisana w całości

4.3. Wyniki ekstrakcji danych

Po dokonaniu ekstrakcji danych z Wikipedii i Wikisłownika dokonano porównania obu zasobów pod kątem użyteczności do ekstrakcji wyrazów wielosegmentowych. Za Wikipedią przemawia fakt, że jej duży rozmiar i obecność wielu nazw i terminów czyni ten zasób odpowiednim do ekstrakcji takich wyrazów wraz z ich definicjami. Wikisłownik zawiera jednak ustrukturyzowane cechy takie jak kategorie gramatyczne i wzorce odmiany, przez co również może być godnym uwagi zasobem. Okazuje się jednak, że prosta analiza ilościowa[5] (tabela 4.2) jednoznacznie pokazuje przewagę Wikipedii i niewielką użyteczność Wikisłownika w kontekście ekstrakcji WW. Mimo to dane wyekstrahowane z Wikisłownika mogą posłużyć np. do ekstrakcji wyrazów spoza słownika CLP, do ekstrakcji etykiet semantycznych oraz niektórych wyrazów wielosegmentowych nieobecnych w Wikipedii. W niniejszej pracy postanowiono jednak skupić się na Wikipedii i wykorzystać jedynie ten zasób.

Tabela 4.2: Ilościowe porównanie danych wyekstrahowanych z Wikipedii i Wikisłownika pod kątem przydatności do ekstrakcji wyrazów wielosegmentowych.

Cecha	**Wikipedia**	**Wikisłownik**
Liczba haseł	1359056	66601
Liczba unikalnych form podstawowych	1227388	40840
Liczba unikalnych wielosegmentowych form podstawowych	973220	1118
Liczba unikalnych wielosegmentowych form podstawowych z co najmniej jednym segmentem rozpoznanym w CLPM	781243	881

[5] Posłużono się wersjami Wikipedii i Wikisłownika ze stycznia 2015 roku.

5. Ekstrakcja etykiet semantycznych dla haseł Wikipedii

Algorytm ekstrakcji etykiet semantycznych został zaprojektowany i zaimplementowany w ramach pracy magisterskiej Autora [10], natomiast później już w ramach przygotowań do pracy doktorskiej został on dopracowany i dostosowany do nowej struktury bazy danych (podrozdział 4.1). Ulepszona wersja została opisana w publikacji [11], po czym jeszcze została ona zmodyfikowana tak, by korzystała ze słownika CLPM. W niniejszym rozdziale przedstawiony zostanie skrócony opis tej metody, z naciskiem na zmiany w stosunku do poprzedniej wersji. Dokładniejszy opis znajduje się w pracach [10, 11].

Celem działania algorytmu jest wyznaczenie **etykiety semantycznej** – krótkiej definicji składającej się z kilku słów, np. dla słowa "Kraków" etykieta powinna brzmieć "miasto", a dla "Karol Bielecki" – "piłkarz ręczny". Etykieta zawiera **rzeczownik główny** oraz inne opcjonalne rzeczowniki lub przymiotniki, jednak powinna być krótka i zwięzła. Czasami trudno jest podać definicję przy pomocy rzeczownika i potrzebne są dodatkowe **operatory**, np. "część samochodu", "rasa kota", "grupa ludzi", które powinny zostać dołączone do etykiety.

5.1. Dane wejściowe i wyjściowe

Wejście. Dla każdego hasła etykieta wyznaczana jest na podstawie odpowiedniego wiersza tabeli `WikiEntry` oraz odpowiadającego mu artykułu `Article`. Atrybuty wykorzystywane przez algorytm to w szczególności nazwa (atrybut *name* tabeli `WikiEntry`), opis (atrybut *desc* tabeli `Article`), a w przypadku przekierowań także nazwa hasła docelowego.

Wyjście. Na wyjściu dla danego hasła oczekujemy etykiety semantycznej. Powinna ona zostać zapisana do bazy danych. W tym celu utworzona została specjalna tabela `Definition`. Poszczególne jej elementy zostaną omówione w opisie algorytmu, natomiast z całą pewnością najważniejszym atrybutem jest *word* – wyraz będący etykietą. Jest to odnośnik do tabeli `Word`, przechowującej wyrazy jedno- i wielosegmentowe. Każdy wyraz posiada atrybut *name* – jego tekstową reprezentację, a także atrybut *tags* – listę oddzielonych spacjami znaczników słownikowych (patrz podrozdział 3.4.1) dla poszczególnych segmentów.

5.2. Ekstrakcja hasła i definicji

Pierwszym etapem ekstrakcji etykiety semantycznej jest podział opisu (atrybut *desc* artykułu) na **segmenty**. Podział na segmenty wykonywany jest w miejscach wystąpień białych znaków i znaków przestankowych. Każdy znak interpunkcyjny jest również uważany za osobny segment. W definicjach encyklopedycznych specjalną rolę pełnią nawiasy – używane są one do wprowadzania informacji uzupełniającej, nie należącej do głównej definicji, dlatego też tekst w nawiasach jest pomijany.

Pierwszy akapit artykułu rozpoczyna się zwykle od powtórzonego hasła – jest to typowa konwencja encyklopedyczna. Aby znaleźć definicję, należy więc wyodrębnić z opisu powtórzone hasło. Niestety, hasło to jest często powtórzone w nieco innej formie – dlatego też stworzono algorytm, który próbuje dokonać podziału kilkukrotnie, za każdym razem pozwalając na mniej restrykcyjne dopasowanie:
Poziom 1. *Pełne dopasowanie* występuje wówczas, gdy na początku opisu występuje dokładnie powtórzone hasło[1]. Poniżej pokazano przykład. Pionowa kreska oznacza **offset**, czyli miejsce, gdzie kończy się rozpoznane powtórzone hasło, a zaczyna się **definicja**. Rozpoznane hasło zostało podkreślone zarówno w nazwie, jak i w opisie.

FIS Team Tour

FIS Team Tour |(znany również jako drużynowy Turniej Trzech Skoczni) - zawody w skokach narciarskich (...)

Poziom 2. *Dopasowanie wszystkich słów* polega na tym, że wszystkie segmenty hasła będące słowami lub liczbami muszą wystąpić w tekście, chociaż ich kolejność może być zmieniona, a między nimi może się znaleźć do 2 innych, nie związanych z nimi segmentów. Pozwala to na akceptację pewnych różnic między hasłem a jego powtórzeniem.

Północ

Rybnik - Północ |- jedna z dzielnic Rybnika, która graniczy od południa z centrum miasta, liczy ok. 8 000 mieszkańców.

Poziom 3. *Dopasowanie skrótów* pozwala na wychwycenie skrótów, które w opisie występują w postaci rozwiniętej.

CB Sevilla

Club Baloncesto Sevilla |(Caja San Fernando) - hiszpański klub koszykarski z siedzibą w Sewilli.

Poziom 4. *Dopasowanie słów podobnych* akceptuje słowa, których odległość edycyjna (Levenshteina) od oryginalnych wynosi mniej niż 20% długości segmentu. Dzięki temu możliwa jest akceptacja alternatywnej bądź niepoprawnej pisowni (błędów ortograficznych).

[1] Dokładnie: atrybut *name* hasła, a w przypadku przekierowania atrybut *name* hasła docelowego.

Francisco Atanasio Dominguez

Francisco Atanasio Domínguez |(ur. około 1740, zm. w 1805), pochodzący z miasta Meksyk franciszkański ksiądz (...)

W przypadku, gdy na żadnym poziomie nie udało się dopasować wszystkich segmentów, dopuszczane jest dopasowanie częściowe. Logowany jest poziom dopasowania oraz ułamek pasujących segmentów, co pozwala na późniejszą analizę takich przypadków. Dodatkowo, jeżeli dla danego offsetu nie udało się znaleźć definicji, podejmowana jest kolejna próba z zerowym offsetem. Dzięki temu np. dla poniższego hasła zostanie wyekstrahowana poprawna etykieta "flaga":

Flaga Ostrołęki

Flaga Ostrołęki składa się z trzech poziomych, równoległych pasów, o równej szerokości i długości.

Niektóre artykuły powinny zostać pominięte, ponieważ nie opisują żadnego obiektu – dlatego też algorytm pomija te hasła, dla których definicji nie udało się wyznaczyć nawet dla próby z zerowym offsetem. Przykładem takiego hasła może być "Lista królów Polski", czy też "Miasta w Europie według liczby ludności".

5.3. Podział definicji na fragmenty zdania

Definicja może składać się z wielu zdań, jednak testy pokazały, że do ekstrakcji etykiety semantycznej wystarczy pierwsze zdanie – dlatego też pozostałe zdania są ignorowane[2]. Wydawałoby się, że etykieta powinna zostać wyekstrahowana z początkowego fragmentu definicji, jednak poniższy przykład pokazuje, że nie zawsze tak jest (podkreślono poprawną etykietę).

Osobowość anankastyczna

Osobowość anankastyczna|, osobowość obsesyjno-kompulsywna - zaburzenie osobowości, w którym występuje wzorzec zachowań zdominowany dbałością o porządek, perfekcjonizmem (...)

Po analizie wielu haseł utworzono heurystyczne reguły generowania kolejnych **fragmentów zdań**, w których wyszukiwana jest etykieta. Wyróżniamy trzy charakterystyczne grupy segmentów:

1. Znaki "—", "–", "-", ":", ",", ".", po których występuje spacja.
2. Formy osobowe czasownika "być" w czasie teraźniejszym bądź przeszłym.
3. Słowo "to".

Każdy fragment zaczyna się od charakterystycznego **prefiksu**. Prefiks zawiera zero bądź więcej segmentów z pierwszej grupy, po których następuje zero bądź więcej segmentów z drugiej grupy, po których

[2]Jest to zgodne ze spostrzeżeniami autorów prac [99, 33].

następuje zero bądź więcej segmentów z grupy trzeciej. W ten sposób można utworzyć zbiór możliwych fragmentów dla danego zdania. Pozostaje pytanie – w którym z nich szukać najpierw? Po przetestowaniu różnych możliwości podjęto decyzję o następującej kolejności:

1. Fragmenty zaczynające się od pauzy "— ", najwyższy priorytet.
2. Fragmenty zaczynające się od myślnika "–".
3. Fragmenty zaczynające się od znaku dywiz "-".
4. Inne fragmenty, najniższy priorytet.

W każdej z powyższych grup porządek ustalany jest według rosnącej odległości od początku zdania.

5.4. Wyszukiwanie rzeczownika głównego

Rzeczownik główny jest podstawowym elementem etykiety, więc musi zostać znaleziony jako pierwszy. W tym celu przeglądane są kolejne segmenty bieżącego fragmentu zdania (z pominięciem prefiksu). Jeżeli prefiks kończył się formą słowa "być" (grupa 2), to poszukujemy rzeczownika w narzędniku (np. "jest gatunkiem japońskiej poezji"), a w przeciwnym wypadku szukamy rzeczownika w mianowniku. Są też wyjątki, np. forma "były" powoduje wyszukiwanie zarówno mianownika ("były piłkarz") jak i narzędnika ("były organizmami"). Wyszukiwanie jest przeprowadzane w oparciu o specyfikację formy (opisaną w podrozdziale 3.4.1), a wynikiem jest pierwszy segment pasujący do specyfikacji, zapisany w postaci znacznika słownikowego. Istnieją przypadki, w których rzeczownik główny mógłby zostać błędnie rozpoznany z powodu niejednoznaczności. Wymagają one stosowania specjalnych reguł heurystycznych[3]. Przykłady podano poniżej,

1. Przymiotnik i rzeczownik, oba w dopełniaczu. Niektóre rzeczowniki mają taka samą formę dopełniacza liczby pojedynczej, jak forma mianownika liczby mnogiej, np. "klasy", "drzewa". Jeżeli taki rzeczownik jest poprzedzony przymiotnikiem w dopełniaczu (np. "średniej klasy"), możemy być pewni, że nie jest on rzeczownikiem głównym.
2. Wyrażenie przyimkowe – nigdy nie zawiera rzeczownika głównego, więc rzeczownik występujący po przyimku powinien być pominięty, np. "z miasta", "w pole".
3. Przymiotnik, a po nim rzeczownik. Czasami występuje związek przymiotnika z rzeczownikiem, przy czym pierwsze z tych słów może też być rzeczownikiem. Przykładowo, w wyrażeniu "włoski malarz" słowo "włoski" mogłoby zostać uznane za rzeczownik główny. Należy wykrywać takie przypadki.

[3] Alternatywą byłoby użycie narzędzia Spejd [8], jednak nie podczas tworzenia algorytmu nie było ono jeszcze opublikowane, a ponadto wydaje się ono nadmiernie złożone w stosunku do rozwiązywanego problemu – konwersja formatu danych byłaby bardziej złożona od samych reguł. Dodatkowo narzędzie to zawęża możliwe interpretacje danego zdania do jednej możliwości – czasami może to spowodować odrzucenie poprawnej struktury zdania.

Tabela 5.1: Relacje i ich operatory. Kolumna "liczba" określa oczekiwaną liczbę rzeczownika głównego po operatorze. Oczekiwany rodzaj to zawsze dopełniacz.

Relacja	Liczba	Przykłady operatorów
Rodzaj	poj. lub mnoga	gatunek, podgatunek, rodzaj, typ, forma, rasa, odmiana
Nazwa	poj. lub mnoga	nazwa, określenie, tytuł, oznaczenie
Część	poj. lub mnoga	część, element, dział, edycja
Zbiór	mnoga	grupa, rodzina, podrodzina, seria, lista, zbiór, gromada

Istnieje też przypadek wymagający zmiany specyfikacji formy – jest nim wyrażenie typu "jeden z ...", "pierwszy z", itp. Przykłady użycia takich wyrażeń to np. "jeden z największych zamków", "największy z zamków", "ostatnie z plemion". W takich przypadkach należy zapamiętać fakt wystąpienia takiego wyrażenia i zmienić oczekiwany przypadek rzeczownika głównego na dopełniacz liczby mnogiej.

5.5. Operatory – pomocnicze elementy etykiety semantycznej

Czasami w miejscu, w którym spodziewamy się rzeczownika głównego, pojawia się słowo pełniące rolę operatora pewnej relacji. Jest tak np. wówczas, gdy opisywany obiekt należy do pewnej kategorii:

Wellnhoferia

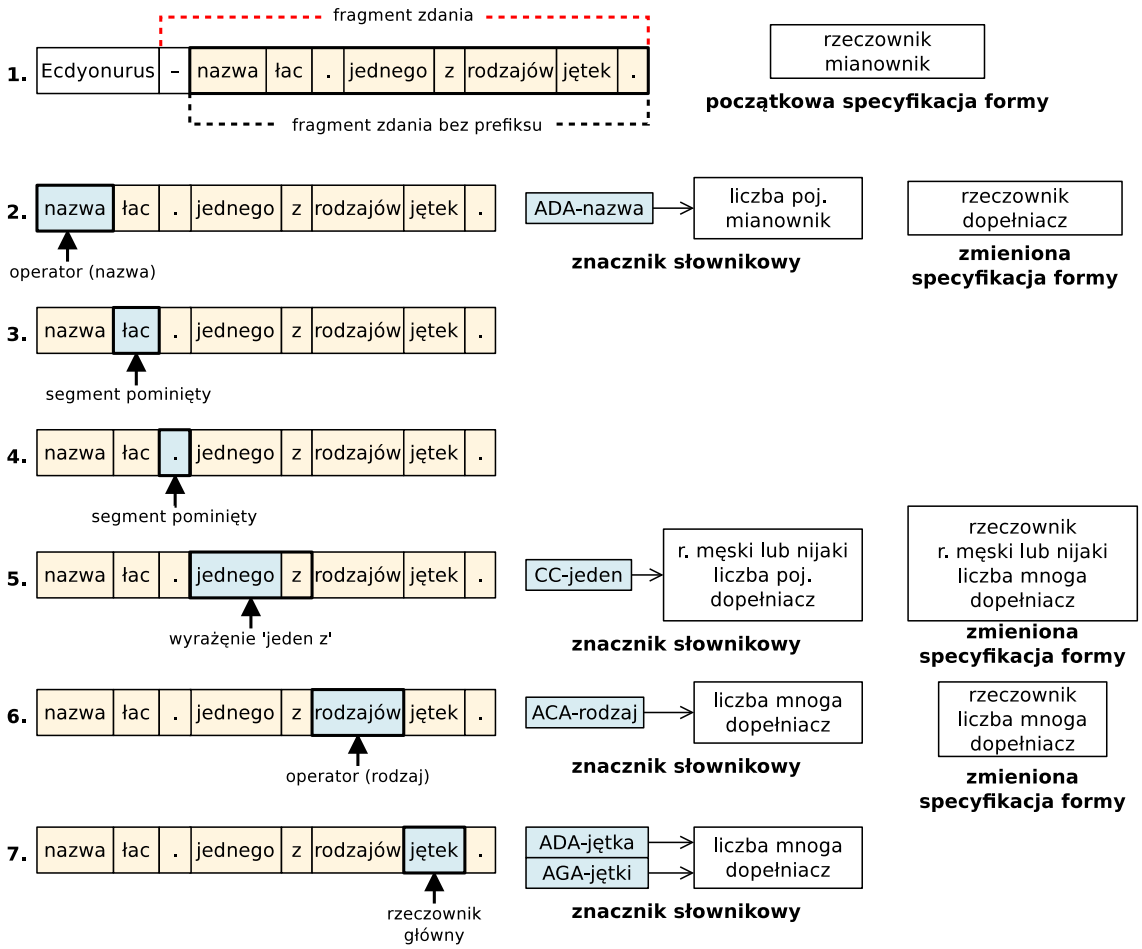
Wellnhoferia - **rodzaj** prehistorycznego ptaka blisko spokrewnionego z archeopteryksem.

W tym przypadku poprawną etykietą jest podkreślone wyrażenie "prehistoryczny ptak", jednak algorytm zatrzyma się na słowie "rodzaj", które uzna za rzeczownik główny. Słowo to jest jednak tylko **operatorem**, który sygnalizuje, że opisywany obiekt jest w pewnej relacji z innym obiektem. Po analizie statystycznej dużej liczby artykułów udało się wyodrębnić wszystkie znaczące operatory. Zostały one zebrane w tabeli 5.1. Po napotkaniu operatora, jest on zapamiętywany jako **pomocniczy segment** etykiety, a specyfikacja formy jest zmieniana do odpowiedniej formy dopełniacza. Przykład rozpoznawania rzeczownika głównego dla skomplikowanego hasła z wieloma operatorami pokazano na rys. 5.1. Specyfikacje formy i znaczniki słownikowe przedstawiono w uproszczonej postaci.

Czasami zdarza się, że rzeczownik będący zwykle operatorem w danym przypadku wyjątkowo pełni jednak formę rzeczownika głównego – w takim wypadku, jeżeli nie uda się znaleźć rzeczownika głównego, algorytm uznaje za rzeczownik główny ostatni z operatorów.

5.6. Pozostałe elementy etykiety semantycznej

Oprócz rzeczownika głównego czasami trzeba włączyć do etykiety dodatkowe segmenty – jest tak np. wówczas, gdy definicja jest wyrazem wielosegmentowym. Poza tym dodanie dodatkowych słów



Rysunek 5.1: Przetwarzanie skomplikowanej definicji z operatorami. **(1)** Powtórzone hasło jest pomijane, a z pierwszego zdania generowany jest pierwszy fragment. Prefiks fragmentu "-" jest pomijany. Początkowa specyfikacja formy oznacza rzeczownik w mianowniku. W następnych krokach, jeżeli bieżący segment pasuje do specyfikacji formy, tworzony jest odpowiedni znacznik słownikowy ograniczony do danej specyfikacji. **(2)** Segment "nazwa" pasuje do specyfikacji i jest na liście operatorów. Specyfikacja formy jest zmieniana na dopełniacz. **(3)** Segment "łac" nie pasuje do specyfikacji. **(4)** Kropka nie jest słowem. **(5)** Rozpoznane zostaje wyrażenie "jednego z", więc oczekiwana forma zmienia się na rodzaj męski lub nijaki (słowo "jednego" może być rodzaju męskiego lub nijakiego), dopełniacz liczby mnogiej. **(6)** Słowo "rodzajów" pasuje do specyfikacji i jest na liście operatorów. Specyfikacja znowu ulega zmianie, tym razem na dowolny rzeczownik w dopełniaczu liczby mnogiej. **(7)** Słowo "jętek" pasuje do specyfikacji i staje się rzeczownikiem głównym. Wynikowy znacznik zawiera dwa identyfikatory: `ADA-jętka` (r. żeński) i `AGA-jętki` (plurale tantum niemęskoosobowy). Ta niejednoznaczność nie może zostać rozwiązana.

(zwłaszcza przymiotników) często uściśla definicję. Z drugiej strony dołączenie nowych segmentów może spowodować, że w etykietach pojawią się zbędne elementy. Po licznych testach utworzono kilka grup segmentów, które mogą zostać dołączone do rzeczownika głównego. Nadano im poniższe oznaczenia:

- **iA**: Spójnik "i", po którym następuje rzeczownik w tej samej formie (liczbie i przypadku), co rzeczownik główny, np. "miasto i gmina".
- **A2**: Rzeczownik w dopełniaczu, np. "ruda żelaza".
- **A**: Rzeczownik w tej samej formie i rodzaju, co rzeczownik główny, np. "lekarz chirurg".
- **C**: Przymiotnik w formie pasującej do rzeczownika głównego, np. "piłkarz ręczny".

Wyszukiwanie dodatkowych segmentów sprowadza się do utworzenia odpowiednich specyfikacji formy dla powyższych przypadków i próby dopasowania segmentów sąsiadujących z rzeczownikiem głównym. Po prawej stronie zachodzi próba dopasowania kolejno wzorców *iA*, *A2*, *A* i *C*[4]. Dodatkowo po lewej stronie podejmowana jest próba dopasowania przymiotnika (*C*) – oznacza to, że dozwolona jest np. etykieta "nieorganiczny związek chemiczny" czy też "wielki mistrz krzyżacki". Oczywiście czasami etykieta może zawierać zbędne elementy, np. "nieduża polana i przełęcz". Okazuje się jednak, że mimo iż słowo "nieduża" nie jest niezbędnym elementem definicji, to pozwala ujednoznacznić formę słowa "polana"[5].

5.7. Zapisywanie etykiet

Wymienione wyżej etapy prowadzą do wyznaczenia etykiety w postaci listy segmentów (znaczników słownikowych) z wyróżnionym rzeczownikiem głównym. Oprócz tego pamiętane są operatory i wyrażenia typu "jeden z". Wyrażenia tego ostatniego typu są usuwane z definicji (np. "jedna z rodzin" → "rodzina")[6]. Dodatkowo etykieta jest sprowadzana do formy mianownika, jeżeli jest w innym przypadku. Na koniec etykieta jest zapisywana zarówno w formie z operatorami ("nazwa rodzaju jętek") jak i bez nich ("jętki").

Podczas analizy danych wyjściowych okazało się, ze często etykieta nie może być wyznaczona poprawnie dla pewnych typów artykułów. Przykładem są strony dotyczące flag i herbów, gdzie zamiast definicji w pierwszym akapicie jest od razu opis danego symbolu. Podobnie, artykuły dotyczące cmentarzy, szkół, szpitali, broni, obszarów geograficznych itp. często nie zawierają definicji. W takim przypadku okazało się, że warto dodać listę ok. 500 ręcznie utworzonych reguł korygujących etykiety w zależności od dopasowania hasła (i opcjonalnie etykiety) do pewnych wyrażeń regularnych. Przykładowo, jeżeli hasło zaczyna się od słowa "Cmentarz", wyznaczana jest od razu etykieta "cmentarz".

Na koniec, jeżeli etykieta jest niepusta, zapisywana jest ona do tabeli `Definition`. Zapisywane atrybuty to: *word* – referencja do tabeli Word zawierającej znaczniki słownikowe etykiety, *simple_word*

[4] Pod uwagę brany jest pierwszy pasujący wzorzec.

[5] Słowo to mogłoby być formą liczby mnogiej wyrazu "polano".

[6] Zwykle nie prowadzi to do utraty danych z wyjątkiem rzadkich przypadków typu "jeden z największych zamków" → "największy zamek".

– to samo lecz dla definicji bez operatorów, *type_l* i *type_r* – typ dodatkowych elementów etykiety (*iA*, *A2*, itp.), *main_index* – pozycja rzeczownika głównego, *first_index* – pozycja właściwej definicji (liczba operatorów).

5.8. Testy i wyniki

Najważniejszy z testów polegał na wybraniu losowej próbki 500 haseł Wikipedii, ręcznym przygotowaniu oczekiwanych etykiet i porównaniu ich z tymi wyznaczonymi automatycznie. Wynik uznawano za pozytywny, gdy etykieta utworzona automatycznie zawierała oczekiwaną – dopuszczano więc zbędne elementy. Wyniki uzyskane przy pomocy słownika CLP podsumowuje tabela 5.2 – spory odsetek błędów wynikających z nieobecności słów takich jak "singel" czy "komputerowy" w CLP pokazuje, że

Tabela 5.2: Wyniki testu poprawności etykiet semantycznych – słownik CLP. Kolumny *liczba niepustych* i *% niepustych* odnoszą się do tych artykułów, dla których udało się znaleźć definicje, a zatem do danych wynikowych z pominięciem haseł odrzuconych.

Kategoria	Liczba	% wszystkich	Liczba niepustych	% niepustych
Poprawne	**452**	**90.4**	**444**	**92.5**
Niepoprawne (przyczyny błędów poniżej)	**48**	**9.6**	**37**	**7.5**
Błędne przekierowanie	1	0.2	1	0.2
Brak definicji	3	0.6	2	0.4
Artykuł nie opisuje żadnego obiektu (powinien zostać pominięty)	5	1.0	5	1.0
Brak słowa w CLP	18	3.6	11	2.2
Inny błąd	21	4.2	18	3.8

Tabela 5.3: Wyniki testu poprawności etykiet semantycznych – słownik CLPM. Pominięte kategorie błędów są bez zmian w stosunku do tab. 5.2

Kategoria	Liczba	% wszystkich	Liczba niepustych	% niepustych
Poprawne	**467**	**93.4**	**459**	**94.3**
Niepoprawne (przyczyny błędów poniżej)	**33**	**6.6**	**28**	**5.7**
Brak słowa w CLPM	3	0.6	2	0.4

wprowadzenie CLPM było uzasadnione. Tabela 5.3 obrazuje poprawę wniesioną przez CLPM.

Wysoka dokładność definicji (94.3% w słowniku wynikowym) pozwala stwierdzić, że algorytm działa bardzo dobrze i może być z powodzeniem wykorzystywany do wzbogacania słowników fleksyjnych o dane semantyczne. Dalsze możliwości rozwoju etykiet semantycznych przedstawiono w podrozdziale 11.2.

6. Problem ekstrakcji wyrazów wielosegmentowych (WW)

Problem automatycznej ekstrakcji wyrażeń wielosegmentowych z tekstu jest rozważany co najmniej od kilkunastu lat – w literaturze anglojęzycznej funkcjonuje pojęcie *multiword expressions* (MWE), które w pracy Saga i in. [86] zdefiniowano jako "idiosynkratyczne interpretacje przekraczające granice słów". W pracy tej wyróżniono 4 kategorie takich wyrażeń dla języka angielskiego. Poniżej przedstawiono ich najbliższe polskie odpowiedniki:

1. Wyrażenia nieodmienne – mają stałe znaczenie, są nieodmienne i semantycznie niedekomponowalne. Przykłady: "ad hoc", "mimo wszystko", "ani mru-mru".
2. Wyrażenia o ustalonej strukturze – funkcjonują jako jedna jednostka słownikowa, posiadająca własne znaczenie i odmieniająca się przez odpowiednie formy gramatyczne. Przykłady: "panna młoda", "biały kruk", "mówić trzy po trzy". Możemy tu zaliczyć również nazwy własne, w tym także wyrażenia uściślające nazwę, np. "stan Michigan", a także inne referencje określonych obiektów, np. "premier Polski". Możemy tutaj wyróżnić wyrazy semantycznie dekomponowalne ("latarnia morska") i niedekomponowalne ("świnka morska"). Niedopuszczalne jest zastępowanie któregokolwiek z segmentów wyrazem bliskoznacznym bądź wstawianie nowych wyrazów między segmenty składowe.
3. Wyrażenia o swobodnej strukturze – jak wyżej, lecz dopuszczają dodawanie lub zamianę niektórych segmentów, a także rozbijanie na części oddzielone innymi segmentami, co nie prowadzi do utraty znaczenia, np. "działać jak płachta na byka", "gotów na czyjeś każde skinienie", "popełnić błąd".
4. Utarte wyrażenia – w pełni dekomponowalne, jednak używane często w danym języku, np. "czyste powietrze", "dookoła świata", "ciężka praca". Znaczenie całego wyrażenia jest prostą sumą znaczeń segmentów, są one więc zaliczone do MWE tylko na podstawie statystyki wystąpień – używa się ich częściej niż innych wyrażeń o podobnym znaczeniu, np. rzadko użyjemy wyrażenia "naokoło świata", chociaż ma znaczenie zbliżone do "dookoła świata".

W niniejszej pracy ograniczono się do wyrażeń pełniących rolę rzeczownika (segmenty składowe niekoniecznie muszą być rzeczownikami). Decyzja ta podyktowana jest głównie faktem, że hasła Wikipedii posiadają taką cechę. Ponadto olbrzymia większość wyrazów wielosegmentowych jest rzeczownikami. Ograniczono się też do **drugiej** kategorii z powyższej listy, ponieważ w pełni odpowiada ona ogólnemu pojęciu wyrazów wielosegmentowych. Pierwsza kategoria została pominięta, ponieważ nie zawiera

Tabela 6.1: Przykłady wyrazów wielosegmentowych, których ekstrakcja jest przedmiotem pracy.

Typ wyrazu	**Przykłady**
Nazwy osób	Józef Piłsudski, Allen Vigneron, Szymon z Wilkowa
Inne nazwy własne	Lazurowa Grota, Polski Związek Wędkarski
Wyrażenia zawierające nazwę	rzeka Carron, jezioro Michigan, premier Polski
Wyrazy pospolite semantycznie niedekomponowalne	panna młoda, świnka morska, czarna dziura
Wyrazy pospolite semantycznie dekomponowalne	chlorek sodu, baza wojskowa, lampa naftowa, zaimek względny

rzeczowników. Kategoria trzecia zawiera głównie związki frazeologiczne nie pełniące roli rzeczownika. Co prawda istnieje grupa wyrażeń takich jak np. "większość absolutna", które posiadają zmienny szyk wyrazów – wyrażenie "absolutna większość" ma takie samo znaczenie. Liczba możliwości jest jednak dokładnie określona i niedopuszczalne jest wymyślanie nowych wariantów, dlatego też poszczególne warianty traktować będziemy w niniejszej pracy jak osobne wyrazy należące do kategorii drugiej[1]. Nie zajmujemy się też w ogóle kategorią czwartą, ponieważ skoro zarówno znaczenie wyrażenia jest sumą znaczeń segmentów, jak i nie ma określonych, sztywnych reguł odmiany, obecność takich wyrażeń w słowniku nie jest niezbędna. Poza tym ciężko byłoby jednoznacznie stwierdzić, czy dane wyrażenie należy do tej kategorii – granica jest tutaj rozmyta[2].

Podsumowując, wyrazy wielosegmentowe (WW) w kontekście niniejszej pracy można zdefiniować jako wyrażenia odmienne, o dokładnie zdefiniowanej, ustalonej strukturze, odmieniające się w całości jak rzeczowniki, pełniące w tekście rolę rzeczowników i posiadające określone, stałe znaczenie. Przykłady takich wyrażeń znajdują się w tabeli 6.1.

6.1. Struktura rzeczownika wielosegmentowego

Wyraz wielosegmentowy pełniący rolę rzeczownika składa się z 2 lub więcej segmentów. Wyróżniamy dwie zasadnicze kategorie segmentów:

– **Segmenty odmienne** tworzą główną część definicji. Mogą nimi być rzeczowniki, przymiotniki, liczebniki lub imiesłowy przymiotnikowe. Segmenty te odmieniają się wraz z całym wyrazem przez przypadki i liczby. W formie podstawowej wszystkie segmenty odmienne występują, podobnie jak cały wyraz, w mianowniku liczby pojedynczej. Wyjątkiem są wyrazy wielosegmentowe *pluralia tantum*, które nie posiadają liczby pojedynczej – wówczas w formie podstawowej cały wyraz, jak i wszystkie segmenty odmienne występują w mianowniku liczby mnogiej. Segmenty odmienne nie

[1] Dzięki wprowadzonym ograniczeniom znikają też problemy z nieciągłymi MWE [29, 38].
[2] Problemy z klasyfikacją takich MWE opisano w pracy [64].

muszą mieć takiego samego rodzaju, np. “kobieta kot”, jednak nie mogą zmieniać rodzaju podczas odmiany.

– **Segmenty nieodmienne** to wszelkie pozostałe segmenty, których forma nie zmienia się niezależnie od formy gramatycznej całego wyrazu. Mogą to być wyrazy odmienne (rzeczowniki, przymiotniki, czasowniki itp.), wyrazy nieodmienne (np. partykuły, spójniki lub wyrazy obcojęzyczne), znaki interpunkcyjne (przecinek, myślnik, kropka, cudzysłów itp.), liczby arabskie bądź rzymskie czy też inne segmenty (np. K2).

Jako przykład rozważmy wyraz “Park Narodowy Gór Stołowych”. Wyraz “Park” jest rzeczownikiem odmiennym, następujące po nim słowo “Narodowy” to odmienny przymiotnik, kolejny wyraz to nieodmienny rzeczownik “Gór” występujący w dopełniaczu, a ostatnim słowem jest nieodmienny przymiotnik “Stołowych”, również w dopełniaczu. Podczas deklinacji tego wyrazu obserwujemy, że segmenty odmienne mają formę identyczną z całym wyrazem, natomiast segmenty nieodmienne pozostają w formie identycznej jak na początku, np. forma narzędnika liczby pojedynczej to “Parkiem Narodowym Gór Stołowych”.

Poszczególne segmenty mogą być oddzielone spacjami lub nie, przykładowo w wyrazie “Park Narodowy Namib-Naukluft” wyróżniamy 5 segmentów – znak dywiz (“-”) uważany jest za osobny segment, nie oddzielony spacjami od sąsiednich. Podczas odmiany musi być zachowana struktura odstępów, np. niedopuszczalną formą jest “Parkiem Narodowym Namib - Naukluft” (dodatkowe spacje) czy też “Parkowi Narodowemu Namib Naukluft” (brak jednego segmentu). Jest to dość restrykcyjne wymaganie, jednak wynika ono z faktu, że pisownia jest jednoznacznie określona przez reguły ortografii.

6.2. Ewaluacja wyników

Aby zweryfikować prawdziwość Tezy 1 przedstawionej we wstępie (podrozdział 1.1), trzeba ocenić jakość wyników generowanych przez algorytmy ekstrahujące WW z tekstu. Typowym podejściem stosowanym do oceny algorytmów przetwarzających tekst w języku naturalnym jest użycie do tego celu specjalnego **korpusu** zawierającego **tagi**, czyli znaczniki poszczególnych wyrazów. Korpus taki dzielimy wówczas na część treningową i testową. Część treningowa służy do uczenia pewnego algorytmu tagującego, którego wyniki są następnie sprawdzane na części testowej – wartości wyznaczonych automatycznie tagów porównywane są z oczekiwanymi. Przykładowo, w taki sposób przeprowadzono testy tagowania częściami mowy (*Part Of Speech tagging*) tekstu w języku polskim [39], czy też testy NER na zbiorze CoNLL-2003 [97]. Takie podejście jest też możliwe dla WW – przykładowo M. Constant i A. Sigogne w swojej pracy [14] użyli takiej metody dla języka francuskiego. Było to możliwe dzięki temu, że dla języka francuskiego istnieje korpus French Treebank posiadający oznakowane WW. Ponieważ jednak nie istnieje korpus języka polskiego otagowany WW[3], w niniejszej pracy podejście takie nie jest możliwe. Ponadto

[3]Korpus NKJP jest otagowany tylko wystąpieniami NE z określonych kategorii [71].

wybór Wikipedii jako źródła danych powoduje, że użyte metody nie będą typowymi statystycznymi klasyfikatorami lub tagerami.

W niniejszej pracy zdecydowano się więc przygotować próbkę testową, tagując ręcznie niewielki korpus testów. Wybrano w tym celu korpus notatek prasowych Polskiej Agencji Prasowej (PAP), zawierający ponad 51 tys. notatek – cały korpus będzie oznaczany w pracy symbolem PAP. Wybrano z niego w sposób losowy 100 notatek prasowych, w których następnie ręcznie oznakowane zostały wyrazy wielosegmentowe. Tagowanie przeprowadzane było przez dwie osoby (autor i promotor pracy), a następnie wyniki były dyskutowane i porównywane aż do osiągnięcia zgody. Powstały korpus oznaczmy jako PAP-WW.

6.2.1. Format korpusu testowego

Każde wystąpienie WW w tekście otoczono podwójnymi nawiasami klamrowymi. Dodatkowo po nawiasach otwierających umieszczona została informacja o odmienności poszczególnych segmentów: gwiazdka ("*") oznacza segment odmienny, a minus ("-") – segment nieodmienny. Przykładowo, zapis `{{** panna młoda}}` oznacza dwusegmentowy WW, w którym oba segmenty są odmienne. Przykładową notatkę prasową oznaczoną wystąpieniami WW przedstawiono poniżej.

```
{{** Mistrz olimpijski}} z Barcelony w {{*--- biegu na 5000 metrów}}
{{** Dieter Baumann}}, ukarany za doping dwuletnim {{*- zakazem startów}}
na {{** arenie międzynarodowej}}, nie może brać także udziału w
{{** zawodach sportowych}} w Niemczech - zadecydował {{** Sąd Najwyższy}}
{{*-- Frankfurtu nad Menem}}.
```

Poszczególne wyrazy wielosegmentowe przeanalizowano w poniższej tabeli.

Wyraz wielosegmentowy	Opis
mistrz olimpijski	Termin sportowy.
bieg na 5000 metrów	Termin sportowy.
Dieter Baumann	Imię i nazwisko – referencja osoby.
zakaz startów	Termin sportowy.
arena międzynarodowa	Wyrażenie idiomatyczne.
zawody sportowe	Termin sportowy.
Sąd Najwyższy	Nazwa instancji sądu.
Frankfurt nad Menem	Nazwa miasta.

Niektóre wyrażenia sprawiały problemy przy tagowaniu. Przykładem mogą być WW, które mają części wspólne, np. w wyrażeniu "premier Wielkiej Brytanii" można dostrzec dwa wyrazy – całe wyrażenie to termin określający szefa rządu państwa brytyjskiego[4], a "Wielka Brytania" to nazwa państwa. W takich przypadkach założono, że trzeba wybrać jedną z możliwości[5] – jest to podyktowane faktem, że wypowiedzi w języku naturalnym konstruujemy tak, by w danym kontekście interpretacja była

[4] Przy tym może on być również użyty w tekście jako referencja konkretnej osoby.
[5] Niedopuszczalne są więc jakiekolwiek wyrażenia mające część wspólną.

jednoznaczna[6]. W omawianym przypadku wybrano całe wyrażenie, ponieważ jest ono tutaj użyte jako jeden wyraz, określający osobę premiera. Nazwa państwa nie pełni tutaj oddzielnej roli semantycznej.

Kolejnym przykładem wątpliwości podczas tagowania jest wyrażenie "na dużą skalę". W rzeczywistości mamy tutaj do czynienia z idiomem. Założono jednak, że interesują nas jedynie wyrażenia pełniące rolę rzeczowników. W takim razie pojawia się pytanie, czy wyrażenie "duża skala" powinno zostać otagowane. Zdecydowano się na nietagowanie, ponieważ w tym przypadku wyrażenie to nie pełni w zdaniu roli semantycznej. Mimo wszystko jednak znacząca większość wyrazów została bezproblemowo otagowana. Wynikowy korpus posiada 572 oznaczone wystąpienia wyrazów wielosegmentowych. Liczba wszystkich segmentów w tekście to 7529.

6.2.2. Przebieg testu

Wejściem testowanego algorytmu ekstrahującego wyrazy wielosegmentowe jest korpus tekstów, natomiast na wyjściu zwraca on korpus otagowany znacznikami w opisanym formacie[7]. Aby zbadać skuteczność algorytmu, należy więc uruchomić go dla nieotagowanej próbki testowej, uzyskując otagowany korpus PAP-TEST. Trzeba teraz porównać znaczniki korpusu PAP-TEST i PAP-WW. W wyniku porównania otrzymujemy cztery zbiory wyrażeń:

- T_i – zbiór poprawnie rozpoznanych wyrażeń, występujących zarówno w PAP-WW, jak i w PAP-TEST z identycznymi tagami.
- T_d – zbiór poprawnie rozpoznanych wyrażeń, występujących zarówno w PAP-WW, jak i w PAP-TEST, lecz różniących się oznaczeniem odmienności segmentów. Przykładem takiego tagu o błędnie rozpoznanej odmienności jest `{{*- arenie międzynarodowej}}` – w tym przypadku słowo "międzynarodowej" oznaczono jako nieodmienne, czyli forma podstawowa brzmiałaby "arena międzynarodowej".
- F_n – zbiór wyrażeń występujących w PAP-WW, lecz nie rozpoznanych w PAP-TEST (wynik fałszywie negatywny).
- F_p – zbiór wyrażeń nie występujących w PAP-WW, błędnie rozpoznanych w PAP-TEST (wynik fałszywie pozytywny).

Pojawia się pytanie: czy elementy zbioru T_d uznać za poprawne wyniki? Ponieważ ciężko znaleźć jednoznaczną odpowiedź, wprowadzono dwa rodzaje testu: **test rozpoznawania** uznaje je za poprawne, natomiast **test ekstrakcji** uznaje je za błędne – podział ten wynika z faktu, że o ile elementy T_d są poprawnie rozpoznane, to jednak nie można ich uznać za w pełni wyekstrahowane WW, ponieważ posiadają błędny wzorzec odmiany.

[6] Inni autorzy prezentują tutaj różne stanowiska, traktując wyrazy składowe jako odrębne wyrażenia [88, 90] bądź nie [68]. Wybrano drugą możliwość, która wydaje się lepiej odzwierciedlać przetwarzanie zdania przez człowieka.

[7] W rzeczywistości na wyjściu otrzymujemy bazę danych wyników, którą można przekształcić do postaci otagowanego korpusu.

Wyniki działania algorytmów rozpoznawania i ekstrakcji informacji z tekstu tradycyjnie podaje się w postaci wartości wskaźników *precision* (P) i *recall* (R). Ich polskie odpowiedniki używane w niniejszej pracy to **precyzja** i **pełność**. Precyzja określa, jaka część rozpoznanych wyników jest poprawna, natomiast pełność – jaką część oczekiwanych wyników rozpoznano poprawnie. Dla testu rozpoznawania wskaźniki te wyrażają się wzorami:

$$P_{rec} = \frac{|T_i \cup T_d|}{|T_i \cup T_d \cup F_p|} \qquad R_{rec} = \frac{|T_i \cup T_d|}{|T_i \cup T_d \cup F_n|} \tag{6.1}$$

Z kolei dla testu ekstrakcji obowiązują wzory:

$$P_{ext} = \frac{|T_i|}{|T_i \cup T_d \cup F_p|} \qquad R_{ext} = \frac{|T_i|}{|T_i \cup T_d \cup F_n|} \tag{6.2}$$

Dla obu metod wyznaczamy jeszcze miarę F_1 (F-measure) będącą ich średnią harmoniczną: $F_1 = \frac{2PR}{P+R}$. Jest to popularnie stosowana miara łącząca precyzję i pełność. Współczynniki F_1 dla obu testów oznaczymy odpowiednio przez F_{rec} i F_{ext}. Ze wszystkich wymienionych miar największe znaczenie będzie dla nas miała wartość F_{rec} i zmaksymalizowanie tej miary będzie celem prac opisywanych w kolejnych rozdziałach.

6.2.3. Znaczenie testu

Użycie opisanej powyżej metody testu ma duże znaczenie – klasyczne testy z podziałem na korpus treningowy i testowy dają zwykle wyniki wyższe niż w późniejszych rzeczywistych zastosowaniach, ponieważ algorytm trenowany jest na danych o takiej samej charakterystyce, co zbiór testowy. Ponadto, oznacza to przyjęcie takich samych kryteriów tagowania dla obu części korpusu. W szczególności, jeżeli korpus treningowy np. nie zawiera pewnych tagów w wyniku błędu, to braku tych tagów będziemy się również spodziewać podczas testów. W przypadku WW oznaczałoby to, że brak pewnych WW w korpusie przełoży się na oczekiwania ich braku podczas testu, co mogłoby zawyżyć wynik. Stosowany w niniejszej pracy test powinien zatem lepiej odzwierciedlać rzeczywiste przypadki użycia, natomiast może on dawać wyniki niższe niż w opisanym podejściu klasycznym. Warto byłoby więc w przyszłości przetestować również inne algorytmy ekstrakcji MWE na zbiorze PAP-TEST. Ponieważ nie istnieją konkurencyjne metody ekstrakcji WW łatwe do przetestowania dla języka polskiego[8], postanowiono nie robić tego w ramach niniejszej pracy – jest to ujęte w planach dalszych badań.

[8] Można by np. przetestować pakiet MWEToolkit [78], jednak wymaga to dużego nakładu pracy.

7. Metoda DM

Ekstrakcja WW w oparciu o słownik utworzony z Wikipedii

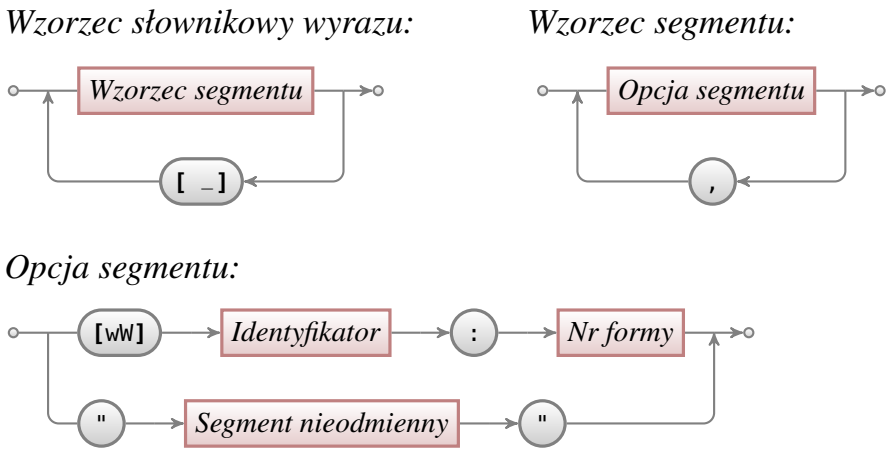
Wyekstrahowane z Wikipedii hasła zapisane w tabeli `WikiEntry` można potraktować jako słownik wyrazów wielosegmentowych. Jest to oczywiście duże uproszczenie: nie wiadomo, które segmenty są odmienne, niektóre z nich mogą być niejednoznaczne, a część haseł nie będzie w ogóle wyrazami wielosegmentowymi. Ponadto Wikipedia zawiera ograniczoną liczbę wyrazów wielosegmentowych. Mimo to z całą pewnością taka prosta metoda może posłużyć jako punkt wyjścia, a zarazem odniesienia (*baseline*) dla metod bardziej zaawansowanych oraz jako składnik ewentualnych metod złożonych. Aby dane wyrażenie (atrybut *name* z tabeli `WikiEntry`) mogło zostać rozpoznane w tekście, potrzebny jest algorytm rozpoznający. Typowe podejście polega na utworzeniu na podstawie hasła **wzorca**, a następnie na rozpoznawaniu tego wzorca w tekście. Do samego rozpoznawania wykorzystano **automat skończony**, który z każdym analizowanym segmentem dokonuje przejścia stanowego. Ten algorytm tworzenia i rozpoznawania wzorców nazwano **DM** (*Dictionary Matching*).

7.1. Wzorce słownikowe

Aby wyraz wielosegmentowy mógł zostać rozpoznany w tekście w różnych formach fleksyjnych, konieczne jest zapisanie informacji o jego sposobie odmiany. Dla języka polskiego opracowano już taki format [38], lecz jest on skomplikowany, ponieważ uwzględnia szeroki zakres cech MWE. Dlatego też zdecydowano się na opracowanie formatu dużo prostszego, uwzględniającego jedynie cechy niezbędne z punktu widzenia niniejszej pracy.

Ponieważ rzeczowniki wielosegmentowe składają się z segmentów odmiennych i nieodmiennych (podrozdział 6.1), potrzebny jest format zapisu tych wyrazów, pozwalający na oznaczenie tych dwóch typów segmentów. O ile segment nieodmienny jest po prostu ciągiem znaków, który musi wystąpić w tekście w dokładnie takiej samej postaci[1], to segment odmienny może odmieniać się na różne sposoby – przykładowo słowo “Kot” może odmieniać się jak nazwa zwierzęcia (biernik l.mn. “Koty”) lub nazwisko (biernik l.mn. “Kotów”). Dlatego też dla segmentów odmiennych potrzebny jest identyfikator i numer formy. W oparciu powyższą obserwację zdefiniowano format zapisu wzorców słownikowych (rys. 7.1).

[1] Ewentualnie na początku zdania może zostać zapisany wielka literą.



Rysunek 7.1: Struktura wzorca słownikowego wyrazu wielosegmentowego. Wyrażenie `[abc]` oznacza, że należy wybrać jeden ze znaków `a`, `b`, `c`.

Wzorzec słownikowy składa się z sekcji dotyczących poszczególnych segmentów, nazywanych **wzorcami segmentów**. Wzorce segmentów oddzielone są spacjami lub znakami podkreślenia, w zależności od tego, czy dane segmenty są oddzielone spacją, czy też nie. Każdy wzorzec segmentu składa się z oddzielonych przecinkami **opcji segmentu** – pozwala to zapisać wzorzec niejednoznaczny, w którym np. nie wiadomo, czy dany segment jest odmienny czy nie. Każda opcja segmentu opisuje segment odmienny lub nieodmienny. Dla segmentu odmiennego występuje mała bądź wielka litera "w" (w zależności od tego, czy słowo jest pisane małą czy wielką literą), a po niej identyfikator, dwukropek i numer formy. Segment nieodmienny jest natomiast po prostu zapisany w cudzysłowie.

W przypadku, gdy wzorzec powstaje w oparciu o kompletną wiedzę o odmianie danego wyrazu wielosegmentowego, każdy wzorzec segmentu zawiera dokładnie jedną opcję segmentu. Poniżej wypisano kilka przykładów takich jednoznacznych wzorców:

Wyraz wielosegmentowy	Wzorzec
hipoteza Sapira-Whorfa	`wADA-hipoteza:1 "Sapira"_"-"_"Whorfa"`
opad atmosferyczny	`wACA-opad:1 wCC-atmosferyczny:8`
Organizacja Jedności Afrykańskiej	`WADA-organizacja:1 "Jedności" "Afrykańskiej"`
Park Narodowy Vanoise	`WACA-park:1 WCC-narodowy:8 "Vanoise"`
rachunek lambda	`wACA-rachunek:1 "lambda"`

7.2. Tworzenie wzorców słownikowych z haseł Wikipedii

Wzorzec słownikowy z hasła Wikipedii można utworzyć na kilka różnych sposobów. W opisywanej metodzie DM wykorzystano algorytm, który zawiera we wzorcu wszystkie możliwe sposoby odmiany, które są zgodne z formą podstawową hasła. Prowadzi to do wielu niejednoznaczności, jednak z drugiej strony nie wymaga żadnych dodatkowych danych i nie wprowadza błędów będących wynikiem każdego sztucznego ograniczenia – jednym z podstawowych założeń niniejszej pracy jest przetestowanie moż-

liwości ekstrakcji wyrazów wielosegmentowych bez wykorzystywania manualnie tworzonych reguł i wzorców.

Sposób tworzenia wzorca dla danego wyrażenia wymaga podziału na segmenty, a następnie utworzenia wzorców segmentu. Podział na segmenty wykonywany jest w miejscach wystąpień białych znaków i znaków przestankowych[2], natomiast tworzenie wzorców segmentu sprowadza się do wygenerowania wszystkich możliwych opcji segmentu zarówno dla przypadku słowa odmiennego, jak i dla nieodmiennego. Tworzenie opcji segmentu odmiennego s przebiega w oparciu o poniższą regułę:

– Dla każdej pary $(id, f) \in tag(s)$[3], jeżeli $label(id)$ zaczyna się od A (rzeczownik), C (przymiotnik) lub D (liczebnik), a oprócz tego numer formy $f \mod 7 = 1$ (mianownik):

  – Gdy pierwsza litera s jest mała lub (s jest pierwszym segmentem, a pierwsza litera $bform(id)$ jest mała), wygeneruj opcję segmentu odmiennego postaci wid:f (litera "w", identyfikator, dwukropek i nr formy).
  – Gdy pierwsza litera s jest wielka, wygeneruj opcję segmentu odmiennego postaci Wid:f.

Z kolei reguła tworzenia opcji segmentu nieodmiennego s wygląda następująco:

– Jeżeli pierwsza litera s jest wielka, wygeneruj opcję segmentu nieodmiennego: "s".

– Jeżeli pierwsza litera s jest mała lub s jest pierwszym segmentem, wygeneruj opcję segmentu nieodmiennego: "s_{lower}", gdzie s_{lower} to słowo s z pierwszą literą zmienioną na małą – notacja ta będzie dalej stosowana w niniejszej pracy.

Przykład. Rozważmy hasło "Droga wojewódzka nr 485". Segment "Droga" może być rzeczownikiem lub przymiotnikiem. Poniżej pokazano znacznik słownikowy tag("Droga") dla tego segmentu w postaci tabeli[4]:

Identyfikator	Znaczenie	Numery form	Opis formy
`ADA-droga`	rzeczownik "droga"	1, 7	M lub W l.p.
`CC-drogi`	przymiotnik "drogi"	15	M, l.p., r.ż.

Forma wołacza zostanie pominięta – zostawiane są jedynie formy mianownika. Kolejny segment "wojewódzka" zostanie jednoznacznie rozpoznany jako przymiotnik. Pozostałe 2 segmenty muszą już być nieodmienne. Tabela 7.1 pokazuje wszystkie utworzone wzorce segmentów.

[2] W niniejszej pracy podział tekstu na segmenty wykonywany jest dla wszystkich algorytmów przy pomocy tej samej funkcji, dokonującej podziału w miejscach wystąpień białych znaków i znaków przestankowych z wyjątkiem wyrażeń z apostrofem typu "Cauchy'ego", które traktowane są jako jeden segment. Podział jest spójny z formatem CLPM, w którym apostrof jest jedynym znakiem interpunkcyjnym dopuszczalnym w formach tekstowych słów. Alternatywnym narzędziem, stosującym jednak inne reguły tokenizacji, jest *Toki* [75].

[3] tag, $label$ i $bform$ to odpowiednie funkcje CLPM (podrozdział 3.4).

[4] Stosujemy oznaczenia przypadków: M, D, C, B, N, Ms, W.

Tabela 7.1: Wzorce segmentów utworzone dla hasła “Droga wojewódzka nr 485”.

Segment	Opcje segmentu	
	odmiennego	nieodmiennego
Droga	`WADA-droga:1`	`"Droga"`
	`wADA-droga:1`	`"droga"`
	`WCC-drogi:15`	
	`wCC-drogi:15`	
wojewódzka	`wCC-wojewódzki:15`	`"wojewódzka"`
nr		`"nr"`
485		`"485"`

Cały wzorzec słownikowy ma postać:

```
"Droga","droga",WADA-droga:1,WCC-drogi:15,wADA-droga:1,wCC-drogi:15 "wojewó
    ↪ dzka",wCC-wojewódzki:15 "nr" "485"
```

Uwagę zwraca fakt wielu nadmiarowych opcji w porównaniu do wzorca zawierającego tylko rzeczywiste możliwości: `wADA-droga:1 wCC-wojewódzki:15 "nr" "485"`. Wyeliminowanie błędnych i nadmiarowych opcji można by przeprowadzić w oparciu o ręcznie tworzone reguły, jednak nie jest to przedmiotem badań niniejszej pracy – w rozdziale 8 przedstawiono metodę na rozwiązanie tego problemu w sposób automatyczny.

7.3. Konstrukcja automatu rozpoznającego wzorce słownikowe

Automat skończony (FSM - *Finite State Machine*) to typowe narzędzie wykorzystywane do rozpoznawania słów i wyrażeń [84]. W przypadku algorytmu DM wybór FSM był więc naturalny. Ogólna idea działania automatu rozpoznającego pojedynczy wzorzec słownikowy jest prosta: dla każdego segmentu przetwarzanego tekstu wyznaczany jest wzorzec segmentu. Na jego podstawie automat powinien wyznaczyć kolejny stan. Osiągnięcie przez automat stanu akceptującego będzie oznaczało rozpoznanie całego wzorca (czyli wystąpienie wyrazu wielosegmentowego w tekście), natomiast brak kolejnego przejścia – odrzucenie danego wyrażenia.

W pierwszej kolejności należy oczywiście zbudować automat rozpoznający wzorce. Celem jest stworzenie **jednego** automatu, który jest w stanie rozpoznawać wiele wzorców jednocześnie – operacja ta musi przebiegać sprawnie nawet dla kilku milionów wzorców. Automat musi pozwalać na rozpoznanie kilku wzorców w jednym stanie, a ponadto jeden wzorzec może być częścią innego, więc z każdym stanem akceptującym musi być powiązany zbiór rozpoznanych wzorców. Zdecydowano się na użycie **automatu Moore’a**, który wyróżnia się tym, że do każdego stanu przypisany jest symbol wypisywany na wyjście w momencie przejścia do tego stanu. Każdemu wzorcowi przyporządkowano identyfikator

liczbowy, a w każdym stanie automat wypisuje na wyjście zbiór identyfikatorów rozpoznanych wzorców. Przy takim założeniu pojęcie stanu akceptującego staje się zbędne.

Konstrukcja automatu rozpoczyna się od automatu pustego, którego zbiór stanów S zawiera jedynie stan początkowy s_0 i nie ma żadnych przejść. Alfabet wejściowy i wyjściowy są na początku puste – będą na bieżąco rozszerzane w miarę dodawania kolejnych wzorców. Następnie dla każdego wzorca wykonywany jest **algorytm dodawania wzorca do automatu** (Algorytm 1).

Algorytm 1: Dodawanie wzorca do automatu.

Wejście : Automat Moore'a ze zbiorem stanów S i stanem początkowym s_0, wzorzec słownikowy $Pattern$

Wyjście : Zmodyfikowany automat rozpoznający wzorzec $Pattern$

1 $PatternSegments \leftarrow$ podziel wzorzec $Pattern$ na segmenty
2 $S_{cur} \leftarrow \{s_0\}$
3 **foreach** P **in** $PatternSegments$ **do**
4 $S_{next} \leftarrow \varnothing$
5 $C_{\varnothing} \leftarrow \{(s,p) \in S_{cur} \times P : \nexists t \in S : s \xrightarrow{p} t\}$
6 **if** $C_{\varnothing} \neq \varnothing$ **then**
7 dodaj nowy stan t_{new} do zbioru S
8 **foreach** $(s,p) \in C_{\varnothing}$ **do** dodaj przejście $s \xrightarrow{p} t_{new}$
9 **foreach** stan $t \in S$ dla którego $\exists (s,p) \in S_{cur} \times P : s \xrightarrow{p} t$ **do**
10 **if** $\exists (s,p) \notin S_{cur} \times P : s \xrightarrow{p} t$ **then**
11 $C_t \leftarrow \{(s,p) \in S_{cur} \times P : s \xrightarrow{p} t\}$
12 dodaj nowy stan t' do zbioru S
13 $output(t') \leftarrow output(t)$
14 **foreach** $(p,u) : t \xrightarrow{p} u$ **do** dodaj przejście $t' \xrightarrow{p} u$
15 **foreach** $(s,p) \in C_t$ **do** usuń przejście $s \xrightarrow{p} t$ i dodaj przejście $s \xrightarrow{p} t'$
16 $t \leftarrow t'$
17 $S_{next} \leftarrow S_{next} \cup \{t\}$
18 **if** P jest ostatnim elementem $PatternSegments$ **then**
19 **foreach** $t \in S_{next}$ **do** $output(t) \leftarrow output(t) \cup \{id(Pattern)\}$
20 $S_{cur} \leftarrow S_{next}$

W pierwszej kolejności (linia 1) wzorzec jest dzielony na segmenty. Krok ten musi zostać objaśniony: polega on na przekształceniu wzorca do listy $PatternSegments$[5], zawierającej jeden element dla każdego segmentu, a także dla znaku oddzielającego segmenty (spacja lub podkreślenie, patrz rys. 7.1). Dla każdego segmentu dodatkowo następuje podział na poszczególne opcje segmentu i w liście $PatternSegments$ na danej pozycji umieszczany jest zbiór możliwych opcji. Dla znaków oddzielających segmenty powstaje zbiór jednoelementowy.

[5] Ponieważ implementacja jest w języku Python, używane jest pojęcie *lista*, a nie *tablica*.

Następnie rozpoczyna się główna pętla algorytmu (linie 3-20), która dokonuje iteracji po elementach P listy $PatternSegments$. W każdej iteracji dany jest zbiór bieżących stanów S_{cur}, który początkowo (linia 2) zawiera jedynie stan początkowy. Podczas działania iteracji wyznaczany będzie zbiór stanów dla następnej iteracji S_{next}, który na końcu (linia 20) stanie się zbiorem bieżącym.

Głównym celem każdej iteracji jest dodanie do automatu przejść dla każdego stanu ze zbioru S_{cur} i każdego elementu zbioru P. Jeżeli co najmniej jedno z tych przejść nie istnieje (linie 5-6)[6], tworzony jest nowy stan (linia 7), a następnie odpowiednie przejścia prowadzące do niego (linia 8)[7].

Aby zachować minimalny rozmiar automatu, konieczne jest wykorzystanie istniejących stanów, dlatego wyznaczane są te stany t, do których prowadzą już istniejące przejścia (linia 9). Jeżeli do danego stanu t prowadzą dokładnie te przejścia, które wynikają ze zbiorów S_{cur} i P, sytuacja jest prosta – można bezpośrednio użyć stanu t, jednak jeżeli istnieją też przejścia nie wynikające ze zbiorów S_{cur} i P (linia 10), należy dokonać podziału stanu t, tworząc jego kopię (linie 11-14), a następnie kierując nowe przejścia wyłącznie do kopii (linia 15).

W końcu, jeżeli P jest ostatnim segmentem wzorca (linia 18), wówczas należy dla każdego stanu $t \in S_{next}$ dodać do zbioru wypisywanego na wyjście automatu $output(t)$ identyfikator rozpoznanego wzorca $id(Pattern)$ (linia 19).

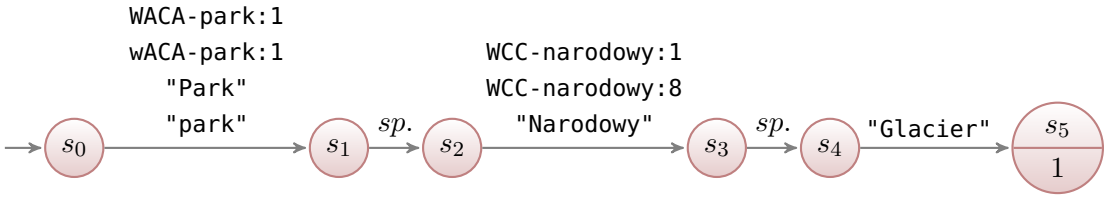
Przykład. Przeanalizujmy proces konstrukcji automatu dla poniższych wyrazów wielosegmentowych:

1. Park Narodowy Glacier
2. Park Decjusza
3. Park Narodowy Glacier Bay

Dla pierwszego hasła utworzony zostanie wzorzec:

```
"Park","park",WACA-park:1,wACA-park:1 "Narodowy",WCC-narodowy:1,WCC-narodowy:8
    ↪ "Glacier"
```

Brak wiedzy o odmianie oraz pisowni (wielka/mała litera dla słowa “park”), a także brak rozpoznawania związków rzeczownika i przymiotnika sprawiają, że wzorzec ten zawiera wiele możliwości. Dodajemy go do pustego automatu, otrzymując diagram stanów przedstawiony na rys. 7.2.



Rysunek 7.2: Diagram stanów po dodaniu wzorca nr 1. Przejścia wielokrotne między daną parą stanów oznaczono dla uproszczenia jedną strzałką. Symbol *sp.* oznacza spację. Liczba pod symbolem stanu oznacza identyfikator rozpoznanego wzorca (wyjście automatu).

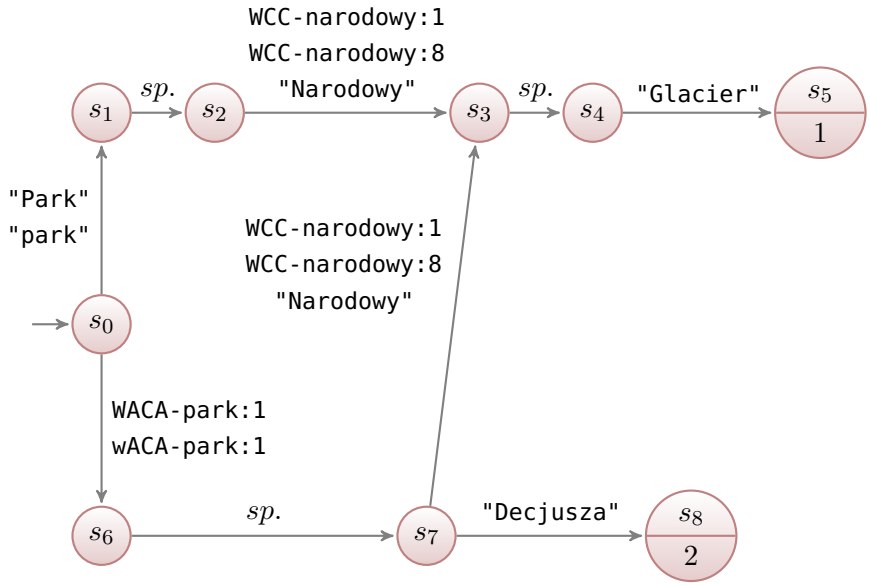
[6] Zastosowano uproszczoną notację przejść: zapis $s \xrightarrow{p} t$, oznacza, że automat może przejść ze stanu s do stanu t, czytając z wejścia symbol p.

[7] Stan ten zostanie dodany do zbioru S_{next} w linii 17.

Operacja dodawania przebiega prosto, ponieważ dla każdej iteracji zbiór stanów jest jednoelementowy i nie ma istniejących przejść (linie 9-17 nie wykonają się ani raz). Na koniec w ostatnim stanie dodajemy identyfikator 1 do zbioru symboli wypisywanych na wyjście (linia 19). Dodajemy teraz drugi wzorzec:

```
WACA-park:1,wACA-park:1 "Decjusza"
```

W tym przypadku możemy zauważyć, że słowo "park" zostało uznane jednoznacznie za odmienne. Jest tak dlatego, ponieważ jest ono jedynym mogącym się odmieniać segmentem w całym wyrażeniu[8]. Powoduje to skomplikowanie diagramu stanów (rys. 7.3).



Rysunek 7.3: Diagram stanów po dodaniu wzorca nr 2.

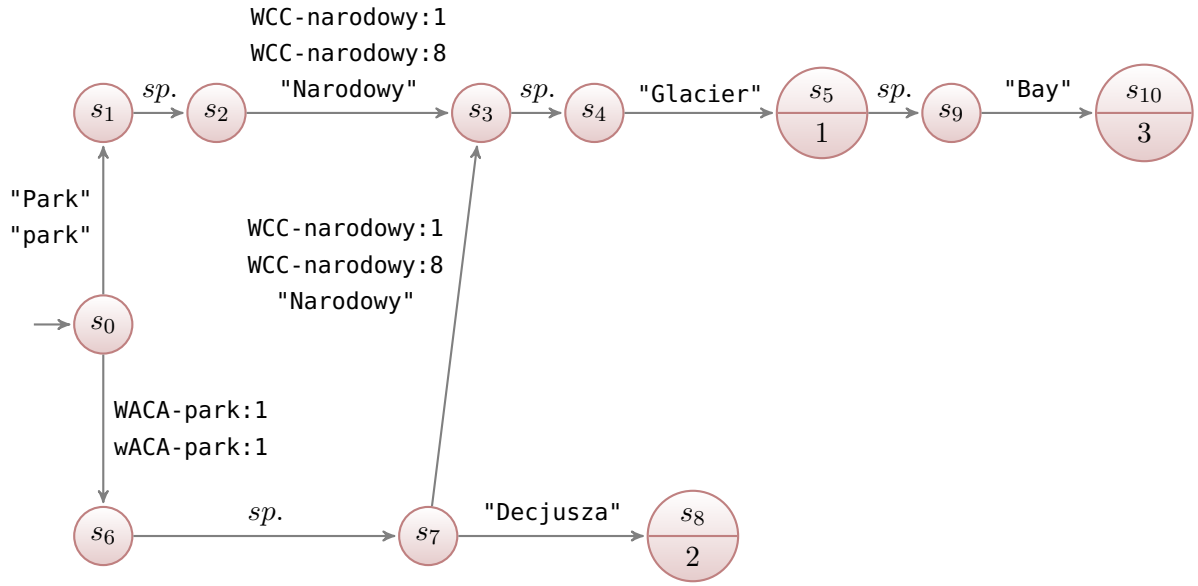
Zauważamy, że już w stanie s_0 dochodzi do sytuacji, w której istniejący stan s_1 musi zostać rozbity (linia 10), ponieważ oryginalnie prowadziły do niego przejścia dla odmiennego i nieodmiennego słowa "park", natomiast we wzorcu nr 2 segment ten jest jednoznacznie odmienny. Utworzony zostaje więc nowy stan s_6. W kolejnej iteracji nastąpi z kolei rozbicie stanu s_2, tworząc stan s_7. Algorytm zapobiega jednak całkowitemu kopiowaniu grafu stanów – w następnej iteracji okazje się, że nie istnieje jeszcze przejście ze stanu s_7 dla wzorca segmentu `"Decjusza"`, więc zostanie utworzony nowy stan s_8, w którym rozpoznany jest wzorzec nr 2.

Ostatni z przykładowych wzorców ma postać analogiczną do wzorca nr 1:

```
"Park","park",WACA-park:1,wACA-park:1 "Narodowy",WCC-narodowy:1,WCC-narodowy:8
    ↪ "Glacier" "Bay"
```

Jedyna różnica polega na dodaniu nowego segmentu na koniec. W takim przypadku algorytm dodawania wzorca będzie wykorzystywał istniejące stany. Co ważne, potrafi on przejść przez istniejące rozgałęzienie i dotrzeć do stanu s_5, w którym rozpocznie się dodawanie nowych stanów (rys. 7.4), prowadzące do utworzenia stanu s_9, a na koniec – s_{10}, w którym rozpoznany jest wzorzec nr 3.

[8] Jest to dodatkowa reguła tworzenia wzorców, nie wspomniana wcześniej ze względów czytelności.



Rysunek 7.4: Diagram stanów po dodaniu wzorca nr 3.

Tak utworzony automat nadaje się już do rozpoznawania wzorców. W rzeczywistości liczba wzorców jest duża (co najmniej kilkaset tysięcy), więc wynikowy automat jest dość skomplikowany. Należy podkreślić, że poświęcono sporo czasu na dopracowanie implementacji opisanego algorytmu tak, aby zminimalizować złożoność pamięciową i czasową, a także pozwolić na sprawną równoległą realizację rozpoznawania wzorców. Utworzono dwa formaty zapisu automatu[9]:

– `DeepMap` – podstawowy format wykorzystujący zagnieżdżone słowniki (`dict`) do reprezentacji stanów i przejść między nimi, zapewniający najefektywniejsze czasowo tworzenie automatu i rozpoznawanie wzorców.

– `MDBMap` – format wykorzystujący bazę danych LMDB tworzoną z istniejącego obiektu `DeepMap`. Zajmuje on znacznie mniej pamięci, co pozwala na równoległe uruchomienie większej ilości procesów rozpoznających wzorce, a także eliminuje konieczność tworzenia automatu po każdym uruchomieniu programu (jest on zapisany na dysku). Szybkość działania jest nieznacznie niższa niż w przypadku `DeepMap`.

Na koniec należy podkreślić, że algorytm tworzenia automatu jest uniwersalny i może zostać użyty do rozpoznawania dowolnych wzorców, które da się przedstawić jako listy zawierające zbiór możliwości dla każdego z segmentów – dlatego też algorytm ten jest jeszcze kilkukrotnie używany w niniejszej pracy[10].

[9] Co prawda dla Pythona istnieją gotowe implementacje automatów skończonych, np. `https://github.com/oxplot/fysom` jednak odrzucono je, ponieważ nie spełniały kryteriów wydajnościowych.

[10] W rzeczywistości algorytm był tworzony pod kątem metody SM (rozdział 9), a dopiero później został zaadaptowany na potrzeby DM.

7.4. Rozpoznawanie wzorców słownikowych w tekście

Wydaje się, że mając już skonstruowany automat rozpoznawanie wzorców staje się proste. Okazuje się jednak, że ponownie problemem stają się różne niejednoznaczności oraz problemy z odmiennością segmentów. Powoduje to konieczność wykorzystania dość złożonego algorytmu również na tym etapie.

W pierwszej kolejności tekst należy podzielić na segmenty, a następnie dla każdego z segmentów należy wygenerować zbiór **wariantów segmentu**, czyli par postaci (P, F), gdzie P to zbiór możliwych opcji segmentu, a F to zbiór numerów form całego wyrazu wielosegmentowego (rzeczownika) odpowiadających opcjom segmentu P. Zbiór F pozwala na sprawdzenie, czy formy poszczególnych segmentów są ze sobą zgodne – dzięki temu możliwe jest wyeliminowanie błędnie odmienionych wyrażeń.

Opisywany algorytm wykorzystuje funkcję $changes(id, f)$, która dla danego identyfikatora jednostki leksykalnej (ze słownika CLPM) id i numeru formy f zwraca zbiór trójek (f_0, g_0, g), gdzie f_0 to numer formy podstawowej dla bieżącego segmentu, g – numer formy całego wyrazu wielosegmentowego, a g_0 – numer formy podstawowej tego wyrazu. Funkcja $changes$ została zaimplementowana poprzez stablicowanie wyników dla wszystkich możliwych wartości argumentów.

Przykład 1. Przeanalizujmy działanie tej funkcji na przykładzie identyfikatora `ABA-kot` i numeru formy 9. Numer ten oznacza dopełniacz liczby mnogiej, czyli odpowiada formie tekstowej “kotów”. Wyznaczmy teraz możliwe wartości f_0: musi być to mianownik, jednak ponieważ forma 9 jest formą liczby mnogiej, forma podstawowa może być liczby pojedynczej lub mnogiej. Mamy więc 2 możliwości: $f_0 \in \{1, 8\}$. Pozostaje teraz stwierdzić, jaka jest możliwa forma podstawowa całego wyrazu wielosegmentowego, zakładając, że bieżący segment jest odmienny. Sprawa jest dość prosta, ponieważ w przypadku rzeczownika forma całego wyrazu musi być taka sama, jak segmentu – dlatego też $g = f$ i $g_0 = f_0$. Wynika stąd, że $changes(\texttt{ABA-kot}, 9) = \{(1, 1, 9), (8, 8, 9)\}$.

Przykład 2. Trudniejszy przypadek występuje wówczas, gdy mamy do czynienia z przymiotnikiem. Dla przykładu rozważmy formę 37 wyrazu `CC-perski`, odpowiadającą dopełniaczowi liczby mnogiej rodzaju niemęskoosobowego, czyli formie tekstowej “perskich”. Ponieważ mamy ponownie do czynienia z liczbą mnogą, jedna z możliwych wartości f_0 to mianownik liczby mnogiej rodzaju niemęskoosobowego, czyli numer 36. Odpowiada mu wartość $g = 9$ (dopełniacz liczby mnogiej) i $g_0 = 8$ (mianownik liczby mnogiej. Nieco bardziej skomplikowane są możliwe formy liczby pojedynczej – rodzaj niemęskoosobowy odpowiada w przypadku przymiotnika kilku możliwościom w liczbie pojedynczej: f_0 może mieć wartość 1 (r. męski żyw.), 8 (r. męski nieżyw.), 15 (r. żeński) lub 22 (r. nijaki). Wszystkim czterem możliwościom odpowiada $g = 9$ i $g_0 = 1$. Podsumowując wymienione możliwości, $changes(\texttt{CC-perski}, 37) = \{(1, 1, 9), (8, 1, 9), (15, 1, 9), (22, 1, 9), (36, 8, 9)\}$.

Mając już omówioną funkcję $changes$ można przystąpić do opisu algorytmu 2 służącego do generowania wariantów segmentu. Podobnie jak przy tworzeniu wzorców, istnieją dwa zestawy reguł: dla segmentu odmiennego i nieodmiennego. Wyniki generowane według obydwu metod są scalane w jeden zbiór. W pierwszej kolejności (linia 1) wyznaczane są opcje segmentu nieodmiennego dla s. Reguły są tutaj podobne, jak przy tworzeniu wzorca słownikowego – powstały zbiór zawiera elementy tworzone

Algorytm 2: Generowanie wariantów segmentu.

Wejście : Segment tekstu s

Wyjście : Warianty segmentu, czyli pary (P, F)

1 $P_0 \leftarrow$ opcje segmentu nieodmiennego dla s
2 $R \leftarrow \{(P_0, F_\infty)\}$
3 **foreach** $(id, F) \in tag(s)$ takie, że $label(id)[0] \in$ { 'A', 'C', 'D' } **do**
4 $\quad C \leftarrow \bigcup_{f \in F} \{(f_0, g) : (f_0, g_0, g) \in changes(id, f)\}$
5 $\quad$ **foreach** f_0, dla którego $\exists (f_0, g) \in C$ **do**
6 $\quad\quad P \leftarrow$ opcje segmentu odmiennego dla (s, id, f_0)
7 $\quad\quad R \leftarrow R \cup \{(P, \{g : (f_0, g) \in C\})\}$
8 **foreach** F, dla którego $\exists P$: $(P, F) \in R$ **do**
9 $\quad P_F \leftarrow \bigcup P : (P, F) \in R$
10 $\quad$ wygeneruj na wyjściu parę (P_F, F)

następująco:

– Jeżeli pierwsza litera s jest mała lub (pierwsza litera s jest wielka i s jest pierwszym segmentem w zdaniu), dodaj element "s_{lower}".

– Jeżeli pierwsza litera s jest wielka, dodaj element "s".

Następnie (linia 2) tworzony jest zbiór wynikowych wariantów segmentu R, zawierający na razie jedynie opcje segmentu nieodmiennego. Symbol F_∞ oznacza specjalny zbiór, do którego należą wszystkie możliwe numery form – sygnalizuje on, że segmenty nieodmienne nie wpływają w żaden sposób na możliwe formy wyrazu wielosegmentowego.

Następuje teraz iteracja po poszczególnych identyfikatorach jednostek leksykalnych wchodzących w skład znacznika słownikowego dla segmentu s (linia 3). Dodatkowo istnieje ograniczenie identyfikatora: musi się on zaczynać od A (rzeczownik), C (przymiotnik) lub D (liczebnik). Dla każdej wartości identyfikatora wyznaczany jest zbiór możliwych par (f_0, g) (linia 4), gdzie f_0 – numer formy podstawowej segmentu s, a g – numer formy (odmienionej) wyrazu wielosegmentowego.

Następnie (linie 5-6) dla każdej możliwej wartości f_0 wyznaczany jest zbiór opcji segmentu odmiennego według poniższych reguł:

– Gdy pierwsza litera s jest mała lub (s jest pierwszym segmentem w zdaniu, a pierwsza litera $bform(id)$ jest mała), dodaj element wid:f_0.

– Gdy pierwsza litera s jest wielka, dodaj element Wid:f_0.

W linii 7 powstałe opcje są dołączane do zbioru R wraz ze zbiorem możliwych form wyrazu wielosegmentowego g. Zbiór R jest jeszcze na koniec optymalizowany poprzez scalanie zbiorów opcji segmentu, dla których zbiory form są identyczne (linie 8-10).

Przykład. Rozważmy opisywany w powyższym przykładzie segment $s =$ “perskich”. W linii 1 zostanie wygenerowana para $(\texttt{"perskich"}, F_\infty)$. Wyznaczmy teraz znacznik słownikowy: $tag(s) = \{(\texttt{CC-perski}, \{30, 32, 34, 37, 41\})\}$. Identyfikator `CC-perski` oznacza przymiotnik, więc wykonujemy linie 4-7. Zbiór C wyznaczamy według poniższej tabeli:

Numer formy f	**Opis formy** f	**Trójki** $changes(f)$
30	D r.męskoos. l.mn.	(1, 1, 9), (29, 8, 9)
32	B r.męskoos. l.mn.	(1, 1, 11), (29, 8, 11)
34	Ms r.męskoos. l.mn.	(1, 1, 13), (29, 8, 13)
37	D r.niemęskoos. l.mn.	(1, 1, 9), (8, 1, 9), (15, 1, 9), (22, 1, 9), (36, 8, 9)
41	Ms r.niemęskoos. l.mn.	(1, 1, 13), (8, 1, 13), (15, 1, 13), (22, 1, 13), (36, 8, 13)

Otrzymujemy $C = \{(1, 9), (29, 9), (1, 11), (29, 11), (1, 13), (29, 13), (8, 9), (15, 9), (22, 9), (36, 9), (8, 13), (15, 13), (22, 13), (36, 13)\}$. Teraz wykonujemy linie 6-7 dla $f_0 \in \{1, 8, 15, 22, 29, 36\}$. W każdym z przypadków generujemy odpowiednie opcje segmentu odmiennego i dodajemy nową parę do zbioru R. Przykładowo dla $f_0 = 15$ utworzymy opcję $p =$ `wCC-perski:15` (tylko jedną, ponieważ s jest pisane małą literą), a do zbioru R dodamy parę $(\{p\}, \{9, 13\})$, ponieważ w zbiorze C znajdowały się pary $(15, 9)$ i $(15, 13)$. Podstępując w ten sposób dla kolejnych wartości f_0 otrzymamy w końcu wynikowy zbiór R, którego zawartość podsumowano w poniższej tabeli.

Opcje segmentu (P)	**Numery form WW** (F)
`"perskich"`	F_∞
`wCC-perski:1`	9, 11, 13
`wCC-perski:8`	9, 13
`wCC-perski:15`	9, 13
`wCC-perski:22`	9, 13
`wCC-perski:29`	9, 11, 13
`wCC-perski:36`	9, 13

Należy jeszcze pogrupować wyniki według zbiorów F:

Opcje segmentu (P)	**Numery form WW** (F)
`"perskich"`	F_∞
`wCC-perski:1`, `wCC-perski:29`	9, 11, 13
`wCC-perski:8`, `wCC-perski:15`, `wCC-perski:22`, `wCC-perski:36`	9, 13

Każdy wiersz tabeli można zinterpretować – np. wiersz ostatni oznacza, że jeżeli słowo “perskich” wchodzi w skład pewnego wyrazu wielosegmentowego i jest odmienne, a ponadto jest rodzaju niemęskoosobowego, to cały wyraz występuje w dopełniaczu (9) lub miejscowniku (13) l. mnogiej, natomiast w formie podstawowej WW segment “perski” występuje w mianowniku liczby pojedynczej (np. “kot perski”, “cyklamen perski”) lub mnogiej (np. “wojny perskie”).

7.4.1. Rozpoznawanie wzorców zaczynających się w danym segmencie

Powstały zbiór wariantów segmentu oznaczmy przez V. Wykorzystywany jest on w algorytmie rozpoznającym wzorce w tekście. W pierwszej kolejności opisany zostanie algorytm pozwalający na rozpoznawanie wzorców zaczynających się od podanego segmentu. Ogólna idea polega na wykorzystaniu utworzonego wcześniej automatu i zbioru V do przejścia przez kolejne stany, zbierając po drodze rozpoznane wzorce aż do momentu, w którym nie istnieją już dalsze przejścia z bieżącego stanu.

Algorytm 3: Rozpoznawanie wzorców dla kolejnych segmentów tekstu.

Funkcja `initialize_matcher()`
  return $\{(s_0, [], F_\infty)\}$

Funkcja `match_segment`(M, V)
  $M_{next} \leftarrow \varnothing$; $R \leftarrow \varnothing$
  foreach $(s, Path, F_{cur}) \in M$ **do**
    $H \leftarrow \{(P, F_{cur} \cap F_{seg}) : (P, F_{seg}) \in V\}$
    foreach $F \neq \varnothing$, dla którego $\exists P : (P, F) \in H$ **do**
      $P_F \leftarrow \bigcup P : (P, F) \in H$
      foreach stan $t \in S$, dla którego $\exists p \in P_F : s \xrightarrow{p} t$ **do**
        $P_t \leftarrow \{p \in P_F : s \xrightarrow{p} t\}$
        $Path_t \leftarrow Path + [P_t]$
        if $\exists u, r : t \xrightarrow{r} u$ **then** $M_{next} \leftarrow M_{next} \cup \{(t, Path_t, F)\}$
        if $output(t) \neq \varnothing$ **then** $R \leftarrow R \cup \{(output(t), Path_t, F)\}$
  scal listy $Path$ w M_{next} dla równych wartości par (t, F)
  return (M_{next}, R)

Dokładny przebieg tej operacji przedstawiono w algorytmie 3. Bieżący stan algorytmu po każdym kroku przechowywany jest zbiorze M trójek $(s, Path, F)$.

- Stan $s \in S$ to stan automatu rozpoznającego wzorce. Początkowa wartość to stan początkowy s_0.
- $Path$ to lista o strukturze takiej samej, jak lista $PatternSegments$ w algorytmie 1, przechowująca ścieżkę dojścia do stanu s, a zarazem rozpoznany (pod)wzorzec słownikowy. Początkowa wartość $Path$ to pusta lista.
- F to bieżący zbiór form WW – na początku jest to F_∞.

Cała procedura rozpoczyna się od wywołania funkcji `initialize_matcher`, która zwraca początkowy zbiór trójek M. Następnie dla każdego kolejnego segmentu tekstu generowany jest zbiór V (algorytm 2) i funkcja `match_segment` wywoływana jest z argumentami M i V. Funkcja ta ma za zadanie wyznaczyć nowy zbiór trójek M_{next} oraz przede wszystkim zbiór rozpoznanych wyników R.

Każdy element zbioru M jest przetwarzany oddzielnie (linia 5). Dla każdego z nich należy teraz wyznaczyć zbiór H, czyli zmodyfikowany zbiór V, w którym zbiory form ograniczono przez F_{cur} (linia 6). Odrzucane są pary (P, F), w których zbiór form jest pusty, a następnie zbiory P są scalane dla równych wartości F (linie 7-8). Pozwala to zmniejszyć złożoność algorytmu – kluczowe jest ograniczenie rozgałęzień i rozpatrywanie stanów łącznie. Połączony dla danego zbioru form F zbiór opcji segmentu oznaczany jest przez P_F. Następnie (linia 9) sprawdzane są po kolei wszystkie stany t, do których ze stanu s istnieje przejście dla opcji segmentu $p \in P_F$ (linia 9). Dla każdego stanu t trzeba teraz wykonać dwie operacje: po pierwsze, ze stanu t mogą istnieć dalsze przejścia – w takim wypadku należy dodać nową trójkę do zbioru M_{next} (linia 12). Po drugie, jeżeli w stanie t rozpoznano jakieś wzorce, odpowiednie wyniki są dodawane do zbioru R (linia 13). Na koniec zbiór M_{next} jest optymalizowany – może się okazać, że dotarto do tego samego stanu z tymi samymi formami dwoma różnymi ścieżkami – wówczas można scalić takie dwie ścieżki $Path$, zmniejszając rozmiar zbioru M_{next} (linia 14). Na koniec zwracamy wartości nowego zbioru M_{next} i wyników R. Zbiór M_{next} zostanie wykorzystany do rozpoznania kolejnego segmentu, natomiast zbiory R są scalane dla poszczególnych iteracji, dając łączny zbiór wyników. Algorytm kończy się, gdy $M = \varnothing$.

Wyniki mają postać trójek $(PatternIds, Path, F)$, gdzie pierwszy element to zbiór rozpoznanych wzorców, drugi – omówiona już wcześniej ścieżka, a trzeci – zbiór numerów form. Taki format wyników pozwala nie tylko na rozpoznanie wzorca, ale także na ekstrakcję informacji o odmianie wyrazu. Temat ten zostanie rozwinięty w kolejnych rozdziałach.

Przykład. Załóżmy, że dany jest przykładowy automat przedstawiony na rys. 7.4. Rozpoznaje on między innymi wyraz wielosegmentowy "Park Narodowy Glacier Bay", przeanalizujmy więc przykład rozpoznawania wzorców w wyrażeniu "Parku Narodowego Glacier Bay". Kolejne kroki działania algorytmu przedstawiono w tabeli 7.2. Opcje segmentu i ścieżki zapisano graficznie – np. ścieżka $[\{\texttt{WACA-park:1}\}, \{spacja\}]$ została zapisana jako `WACA-park:1` . Możemy zauważyć, że dla separatora (*spacja*) również generowany jest zbiór V – reguła generowania jest formalnie przedstawiona w kolejnym podrozdziale.

Widać, jak wczytywanie poszczególnych segmentów powoduje przejście przez kolejne stany automatu z jednoczesnym ograniczaniem zbioru form – po przeczytaniu słowa "Parku" mamy 3 możliwości, jednak słowo "Narodowego" ogranicza je do jednej. Pełną zawartość wynikowego zbioru R pokazuje tab. 7.3.

7.4.2. Rozpoznawanie wzorców w tekście

W poprzednim podrozdziale omówiono rozpoznawanie wzorców słownikowych zaczynających się od danego segmentu, jednak celem pracy jest rozpoznawanie wyrazów wielosegmentowych w dowolnym tekście. Aby rozpoznać wzorce zaczynające się od dowolnego segmentu, należy wykonać algorytm 3 zaczynając od kolejnych segmentów tekstu. Sposób realizacji pokazuje algorytm 4.

Algorytm opiera się na leniwym generowaniu segmentów z tekstu (linia 2), aby zminimalizować złożoność pamięciową. Dla każdego segmentu lub separatora[11] wykonywana jest iteracja głównej pętli. Zbiór

[11] Jest nim spacja lub znak podkreślenia, gdy między segmentami nie ma spacji.

Tabela 7.2: Kolejne kroki działania algorytmu rozpoznającego wzorce dla wyrażenia “Parku Narodowego Glacier Bay” przy założeniu, że wyrażenie, to nie rozpoczyna zdania. Zawartość zbioru R podano w uproszczeniu – ograniczono ją do identyfikatorów wzorca.

Segment/sep.	**Zbiór wariantów** V	**Zbiór** M_{next}	R
Parku	(`"Parku"` , F_∞), (`WACA-park:1` , $\{2,6,7\}$)	(s_6, `WACA-park:1` , $\{2,6,7\}$)	$\varnothing$
spacja	(`␣` , F_∞)	(s_7, `WACA-park:1␣` , $\{2,6,7\}$)	$\varnothing$
Narodowego	(`"Narodowego"` , F_∞) (`WCC-narodowy:1` , $\{2,4\}$) (`WCC-narodowy:8` , $\{2\}$)	(s_3, `WACA-park:1` `WCC-narodowy:1,WCC-narodowy:8` , $\{2\}$)	$\varnothing$
spacja	(`␣` , F_∞)	(s_4, `WACA-park:1` `WCC-narodowy:1,WCC-narodowy:8␣` , $\{2\}$)	$\varnothing$
Glacier	(`"Glacier"` , F_∞)	(s_5, `WACA-park:1` `WCC-narodowy:1,WCC-narodowy:8` `"Glacier"` , $\{2\}$)	1
spacja	(`␣` , F_∞)	(s_9, `WACA-park:1` `WCC-narodowy:1,WCC-narodowy:8` `"Glacier"␣` , $\{2\}$)	1
Bay	(`"Bay"` , F_∞)	$\varnothing$	1, 3

Tabela 7.3: Elementy końcowego zbioru wyników R.

Id. wzorca	**Ścieżka** ($Path$)	**Formy** (F)
1	`WACA-park:1` `WCC-narodowy:1,WCC-narodowy:8` `"Glacier"`	2
3	`WACA-park:1` `WCC-narodowy:1,WCC-narodowy:8` `"Glacier"` `"Bay"`	2

$Matchers$ przechowuje bieżący stan wszystkich procesów rozpoznawania wzorców – jego elementami są zbiory M przedstawione w algorytmie 3. Dla separatora generowany jest uproszczony zbiór wariantów (linia 5), natomiast dla segmentów po wygenerowaniu zbioru wariantów zgodnie z algorytmem 2 (linia 7) do zbioru $Matchers$ dodawany jest nowo zainicjowany stan operacji rozpoznawania wzorców[12]. Teraz wykonywany jest kolejny krok każdej instancji procesu rozpoznawania wzorców (linia 9). Wyniki są dodawane do zbioru R (linia 11), natomiast nowy stan trafia do zbioru $Matchers_{next}$ tylko wówczas, gdy jest on niepusty (linia 12). Na koniec zbiór $Matchers$ jest aktualizowany (linia 13).

W rzeczywistości cały proces rozpoznawania jest nieco bardziej skomplikowany – np. wykrywanie, czy segment zaczyna nowe zdanie wymaga przechowywania dwóch poprzednich segmentów. Pominięto jednak takie dodatkowe informacje, ponieważ skomplikowałyby one znacznie opis algorytmów.

[12]Nie ma potrzeby rozpoczynania procesu rozpoznawania dla separatorów, ponieważ wyrazy wielosegmentowe nie mogą zaczynać się od separatora.

Algorytm 4: Rozpoznawanie wzorców słownikowych w tekście.

Wejście : Tekst $Text$, automat rozpoznający wzorce
Wyjście : Zbiór wyników R

$R \leftarrow \varnothing$; $Matchers \leftarrow \varnothing$
foreach s **in** generuj leniwie segmenty i separatory tekstu $Text$ **do**
  $Matchers_{next} \leftarrow \varnothing$
  if s jest separatorem **then**
    $V \leftarrow \{(\{s\}, F_\infty)\}$
  else
    $V \leftarrow$ wygeneruj warianty segmentu s
    $Matchers \leftarrow Matchers \cup \{$`initialize_matcher()`$\}$
  foreach $M \in Matchers$ **do**
    $(M_{next}, R_{cur}) \leftarrow$ `match_segment(`M, V`)`
    $R \leftarrow R \cup R_{cur}$
    if $M_{next} \neq \varnothing$ **then** $Matchers_{next} \leftarrow Matchers_{next} \cup \{M_{next}\}$
  $Matchers \leftarrow Matchers_{next}$

7.5. Przetwarzanie i zapis wyników

Zbiór wyników R zawiera rezultaty, które można teraz przetwarzać dalej w zależności od potrzeb. W niniejszej pracy zdecydowano, że wyniki są zapisywane do bazy danych w rozbudowanej postaci, dzięki czemu mogą stanowić dane wejściowe kolejnych algorytmów lub trafić do słownika.

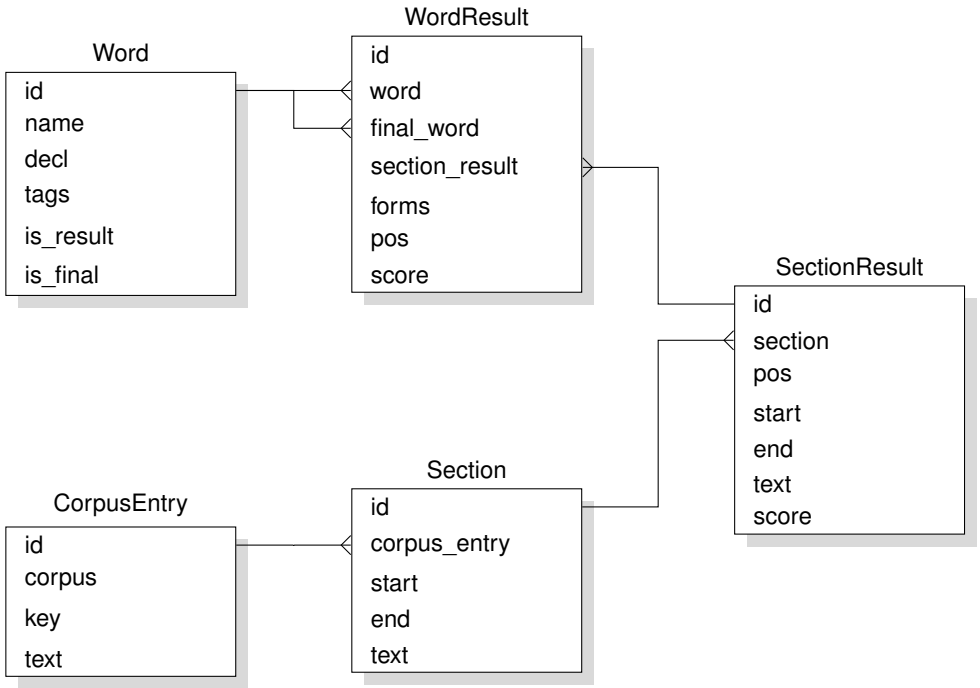
7.5.1. Format zapisu wyników

Na rysunku 7.5 przedstawiono docelową strukturę bazy danych. Tabela `CorpusEntry` przechowuje wejściowy korpus, na którym dokonywane jest rozpoznawanie wzorców. W przypadku korpusu PAP jest on podzielony na poszczególne notatki. Każda z nich ma swój klucz przechowywany w atrybucie *key*. Atrybut *corpus* zawiera nazwę korpusu – w tym przypadku "pap", natomiast *text* to tekstowa zawartość notatki sformatowana tak, że jedyne dozwolone białe znaki to spacja i znak nowego wiersza oddzielający akapity.

Pozostałe tabele służą już do przechowywania wyników. Pierwsza z nich, `Section`, przechowuje **sekcje** zawierające wyniki rozpoznawania wzorców – dla każdej grupy wyników mających części wspólne tworzona jest jedna sekcja. Przykładowo, w wyrażeniu "Papieża Jana Pawła II" mogą zostać rozpoznane ciągi segmentów "Papieża Jana Pawła", "Jana Pawła", "Jana Pawła II" i "Pawła II" – wówczas wszystkie one zostaną zakwalifikowane do jednej sekcji. Każda sekcja odnosi się do konkretnej notatki (atrybut *corpus_entry*), a zakres segmentów, których dotyczy, zapisany jest w postaci indeksów *start* i *end*. Dodatkowo pole *text* zawiera tekstową reprezentację całej sekcji.

Każdy posiadający rozpoznane wzorce ciąg segmentów w obrębie sekcji nazywany jest **wynikiem sekcji** i przechowywany jest w osobnym wierszu tabeli `SectionResult`. Wynik sekcji oprócz referencji do swojej sekcji posiada, podobnie jak cała sekcja, atrybuty *start*, *end* i *text*. Pojawia się jednak pytanie, co zrobić w przypadku gdy dla jednej sekcji mamy kilka możliwych wyników. Ponieważ tworząc test założono, że nie pozwalamy na to, by wyniki nachodziły na siebie, potrzebna jest metoda pozwalająca na wybór dla każdej sekcji najlepszego podzbioru wyników nie posiadającego części wspólnych. Pierwszą regułą, która po zastosowaniu dała od razu dobre wyniki, jest zasada wykluczania wszystkich wyników, których lista segmentów jest częścią listy segmentów innego wyniku. Przykładowo spośród wyników "Parku Narodowym Glacier" i "Parku Narodowym Glacier Bay" wybrany zostanie tylko ten drugi. Pozostaje jednak pytanie, co zrobić w przypadku wyników pokrywających się częściowo. Aby umożliwić obsługiwanie takich przypadków, wprowadzono pojęcie **oceny** każdego wyniku (atrybut *score*), będącej liczbą. Jeżeli dla każdego wyniku znana jest wartość oceny, wówczas można wybrać dla danej sekcji taki podzbiór wyników nie posiadających części wspólnej, aby sumaryczna ocena była jak największa. Problem wyznaczania ocen zostanie przedstawiony później. W oparciu o powyższą klasyfikację wyznaczana jest wartość atrybutu pozycji (*pos*): w obecnej implementacji możliwe wartości pozycji to 0 (wynik został zaakceptowany) lub 1 (wynik odrzucony).

O ile dla samego przeprowadzenia testu rozpoznawania (patrz podrozdział 6.2.2) powyższe dane są wystarczające, to test ekstrakcji wymaga jeszcze przechowywania informacji o tym, które segmenty wyrazu wielosegmentowego są odmienne. Ponadto wyniki ekstrakcji mogą służyć nie tylko do testów, ale także np. do tworzenia słownika wyekstrahowanych wyrazów. Dlatego też potrzebna jest dodatkowa tabela `Word`, przechowująca **wyrazy** wielosegmentowe. Dla każdego wyrazu przechowywana jest jego



Rysunek 7.5: Struktura bazy danych wyników rozpoznawania wzorców słownikowych (DM).

reprezentacja tekstowa (atrybut *name*), informacja o tym, które wyrazy są odmienne (atrybut *decl*, łańcuch znaków zawierający gwiazdkę "*" dla segmentu odmiennego i minus "-" dla nieodmiennego), lista znaczników słownikowych rozpoznanych segmentów odmiennych (*tags*), a także atrybut *is_result* = `true`, oznaczający, że wyraz utworzony został z wyników[13]. Para atrybutów $(tags, decl)$ nosi nazwę **wzorca odmiany**, ponieważ w pełni determinuje sposób odmiany wyrazu wielosegmentowego.

Pozostała do omówienia jeszcze jedna ważna tabela przechowująca **wyniki wyrazu** – `WordResult`, która łączy ze sobą wyniki sekcji i wyrazy relacją wiele-do-wielu – każdemu wynikowi sekcji może odpowiadać kilka możliwych wyrazów, natomiast każdy wyraz może wystąpić w wielu wynikach sekcji. Atrybuty `WordResult` to referencje do wyrazu (*word*) i wyniku sekcji (*section_result*), a także zbiór numerów form danego wystąpienia (*forms*). Dodatkowo, ponieważ podczas testu trzeba będzie wybrać jedną spośród wielu możliwości odmiany danego wyrazu, dodano atrybut liczbowej **oceny** (*score*) oraz pozycji (*pos*) wynoszącej 0 bądź 1, podobnie jak dla wyniku sekcji. W oparciu o ocenę najlepszego wyniku wyrazu wyznaczana jest ocena wyniku sekcji. Pozostaje pytanie – jak ocenić liczbowo wynik wyrazu? Gdyby każdy rozpoznany wzorzec słownikowy posiadał pewną wartość oceny, można by je zsumować. W przypadku metody DM ciężko jednak o dobrą funkcję oceniającą, dlatego też przyjęto, że wszystkie oceny wynoszą 1 – wybór jest więc losowy[14]. Dla kolejnych metod (np. SM – rozdział 9) oceny są już różne w zależności od wzorca. Wynik wyrazu ma jeszcze atrybut *final_word*. Został on utworzony na potrzeby tworzenia słownika wyrazów wielosegmentowych i zostanie omówiony później.

7.5.2. Proces przetwarzania i zapisu wyników

Zapis danych w powyższym formacie na podstawie zbioru wyników R jest dość złożonym zadaniem. Trudność polega nie tylko na istnieniu wielu relacji jeden-do-wielu między poszczególnymi tabelami, ale także na tym, że elementy zbioru R w rzeczywistości są w relacji wiele-do-wielu z wynikami wyrazu (`WordResult`). Jest tak dlatego, ponieważ rozpoznany wzorzec często jest niejednoznaczny (dany segment może niejednoznaczny: odmienny lub nie), więc może z niego powstać wiele wyników wyrazu, a z drugiej strony kilka elementów zbioru R może dać w wyniku ten sam wynik wyrazu (choćby wówczas, gdy dany wzorzec jest rozpoznany w dwóch różnych stanach automatu, a także wtedy, gdy wzorce o jednakowej długości mają część wspólną[15]).

Rozwiązaniem jest więc rozbicie każdego elementu zbioru R na składowe według wzorca odmiany i grupowanie tych składowych, tworząc strukturę bazy danych. Dane są zapisywane do bazy dopiero po przetworzeniu całej notatki (`CorpusEntry`) – ma to również na celu optymalizację zapisu do bazy danych.

Przykład. Proces przetwarzania i zapisu wyników zostanie pokazany na przykładzie zbioru R przedstawionego w tab. 7.3. Dodatkowo załóżmy, że treść rozpoznawanej notatki (atrybut *text* w tabeli `CorpusEntry`) to "Pojechałem do Parku Narodowego Glacier Bay." – wyniki nie zmienią się. W pierwszej kolejności utwórzmy wszystkie możliwe wyrazy (`Word`). W naszym przypadku sytuacja jest

[13] Tabela `Word` jest też używana np. do przechowywania etykiet semantycznych – podrozdział 5.7.
[14] W przyszłości można zastosować jakąś heurystykę oceniającą.
[15] W algorytmach innych niż DM dochodzi do tego często.

prosta, ponieważ istnieje dokładnie jedna forma podstawowa: mianownik liczby pojedynczej. Gdyby wyraz wystąpił w liczbie mnogiej, mogłaby się pojawić niejednoznaczność. Ponadto dzięki temu, że przypadek jest inny od mianownika, udało się zidentyfikować jednoznacznie segmenty odmienne i dla każdego rozpoznanego wzorca mamy dokładnie po jednym WW. Jedyne, czego nie udało się ujednoznacznić, to typ rodzaju męskiego segmentu "Narodowy"[16]. Poniżej przedstawiono wynikową tabelę `Word`.

id	name	decl	tags
1	Park Narodowy Glacier	`**-`	`ACA-park:1 CC-narodowy:1,8`
2	Park Narodowy Glacier Bay	`**--`	`ACA-park:1 CC-narodowy:1,8`

W tabeli `Section` znajdzie się tylko poniższy wiersz[17].

id	start	end	text
1	2	6	Parku Narodowego Glacier Bay

Sekcji 1 odpowiadają dwa wyniki sekcji (tabela `SectionResult`), z których tylko drugi jest brany pod uwagę do testu, ponieważ pierwszy zawiera się w nim.

id	section	pos	start	end	text	score
1	1	1	2	5	Parku Narodowego Glacier	1
2	1	0	2	6	Parku Narodowego Glacier Bay	1

Ponieważ wyrazy są jednoznaczne, a każdy wzorzec rozpoznano tylko w jednym miejscu, tabela `WordResult` zawiera również 2 wiersze:

id	word	section_result	pos	forms	score
1	1	1	0	2	1
2	2	2	0	2	1

Rozważony przypadek był celowo prosty, by łatwo było zrozumieć podstawowy mechanizm zapisu wyników.

7.6. Testy metody DM

Wzorce słownikowe do testu metody DM utworzono ze wszystkich wielosegmentowych haseł Wikipedii[18], dla których co najmniej jeden segment może być odmienny. Liczba utworzonych wzorców to 666853. Następnie utworzono automat rozpoznający te wzorce i zapisano w formacie `MDBMap`. Automat ten posłużył do testu wykonanego na korpusie testowym PAP-WW. Wynik testu pokazano w pierwszym wierszu tabeli 7.4.

[16] Potrzebne byłyby heurystyczne reguły wynikające z gramatyki języka polskiego, których wprowadzania na tym etapie chcemy uniknąć. Pomóc mogłyby też tagery, lecz wprowadziłyby one pewien odsetek błędów.

[17] Segmenty numerujemy od zera, a prawy koniec nie wchodzi do przedziału – stąd indeksy 2 i 6.

[18] Przez hasło rozumiemy atrybut *name* z tabeli `WikiEntry`.

Tabela 7.4: Wyniki testów rozpoznawania i ekstrakcji wyrazów wielosegmentowych metodą DM.

	Test rozpoznawania			**Test ekstrakcji**		
Źródło wzorców	P_{rec}	R_{rec}	F_{rec}	P_{ext}	R_{ext}	F_{ext}
1. Wszystkie hasła Wikipedii	80.97	**42.54**	**55.78**	58.71	30.85	40.44
2. Hasła Wikipedii z etykietą sem.	81.54	41.19	54.73	59.06	29.83	39.64
3. Hasła rozpoznane heurystycznie	**83.28**	40.51	54.50	**78.75**	38.31	51.54
Metoda łączona: 3 + 1	80.71	42.54	55.72	74.92	**39.49**	**51.72**

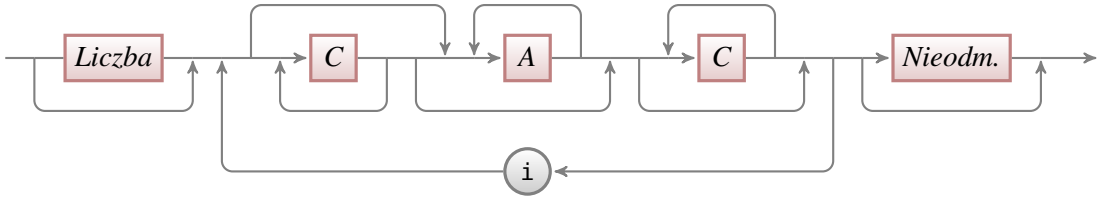
Zgodnie z przewidywaniami okazuje się, że poziom precyzji P_{rec} jest wysoki przy jednocześnie niskiej wartości pełności R_{rec}. Test ekstrakcji pokazuje natomiast główną słabość algorytmu – w przypadku istnienia kilku alternatywnych wyników z różnymi wzorcami odmiany wybierany jest losowy z nich, co znacząco pogarsza statystykę poprawnie rozpoznanych segmentów odmiennych – stąd niska wartość F_{ext}. Widać wyraźnie, że o ile hasła Wikipedii mogą posłużyć do utworzenia słownika wyrazów wielosegmentowych o dość wysokiej jakości haseł, to słownik taki nie będzie kompletny – istnieje potrzeba znalezienia metody, która osiągnie dużo wyższy wskaźnik pełności. Metody takie zostaną przedstawione w kolejnych rozdziałach.

Celem niniejszej pracy jest wprowadzenie automatycznych metod ekstrakcji wyrazów wielosegmentowych, które nadają się do późniejszego wzbogacenia o różne reguły, heurystyki i klasyfikatory. Postanowiono jednak juz teraz wykonać kilka eksperymentów z heurystykami w tych przypadkach, gdy były one łatwe do przeprowadzenia. Kolejne wiersze tabeli 7.4 dokumentują takie eksperymenty.

Metoda nr 2 polega na ograniczeniu haseł tylko do tych, dla których udało się wyznaczyć etykiety semantyczne (rozdział 5). Podczas prac nad etykietami semantycznymi zauważono, że hasła bez etykiet bardzo często są jedynie tytułami pewnych tabel i zestawień, np. “Lista miast Polski”. Dlatego też eliminacja takich haseł powinna pozostawić rzeczywiste wyrazy wielosegmentowe. Okazuje się jednak, że zysk precyzji jest niewielki[19], a pełność spada ze względu na fakt, że dla niektórych wyrazów nie udało się wyznaczyć etykiety. Wariant ten spisuje się więc nieco gorzej.

Odmienna sytuacja występuje dla metody nr 3. W tym przypadku wykorzystano utworzony na potrzeby pracy magisterskiej heurystyczny algorytm wyznaczający wzorzec odmiany wyrazu wielosegmentowego (rozdział 2.6.1 pracy [10]). Opiera się on na obserwacji, że większość haseł ma strukturę przedstawioną na rys. 7.6: Na pierwszej pozycji występuje opcjonalnie liczba rzymska lub arabska, po której występuje ciąg części odmiennych oddzielonych spójnikami “i”. Każda z tych części składa się z opcjonalnych przymiotników (*C*), po których występują opcjonalne rzeczowniki (*A*) i znowu opcjonalne przymiotniki. Na koniec może wystąpić jeszcze ciąg segmentów nieodmiennych. W każdym wyrazie musi oczywiście wystąpić co najmniej jeden segment odmienny. Przykłady wyrazów wielosegmentowych pasujących do tej struktury (segmenty odmienne podkreślono) to “Wojewódzka i Miejska Biblioteka Publiczna w Zielonej Górze” i “115 Pułk Piechoty im. Księcia Williama”. Najważniejszą zaletą stosowania tej metody jest

[19] Jest to najprawdopodobniej spowodowane niewielką szansa wystąpienia wyrażeń typu “Lista miast Polski” w tekście.



Rysunek 7.6: Heurystycznie wyznaczona oczekiwana struktura wyrazu wielosegmentowego.

poprawne rozpoznanie segmentów odmiennych – pokazują to wyraźnie wyniki. Jednocześnie dochodzi jednak do spadku R_{rec}, ponieważ istnieją poprawne WW nie pasujące do przedstawionego wzorca.

Na koniec przedstawiono jeszcze wyniki metody będącej złożeniem metody 3 i 1: wyrazy, które nie zostaną rozpoznane przez metodę 3, są rozpoznawane metodą 1. Podejście takie powoduje zachowanie F_{rec} z metody 1. przy jednoczesnym zwiększeniu F_{ext}. Eksperymenty z prostą heurystyką pokazują, że wyniki metody DM można poprawiać stosując ręcznie tworzone reguły odmiany – warto więc w przyszłości kontynuować badanie tego rodzaju ulepszeń, być może wykorzystując bardziej złożone narzędzia regułowe, np. Spejd [8] i SProUT [66]. Wydaje się jednak, że lepszym rozwiązaniem jest wyznaczenie wzorców odmiany haseł w sposób automatyczny – zagadnieniu temu poświęcony jest kolejny rozdział.

8. Metoda pDM

Ekstrakcja WW z wykorzystaniem wzorców odmiany

Zaprezentowane wyniki metody DM pokazują, że użycie heurystycznej metody określającej sposób odmiany wyrazów może dawać całkiem dobre efekty. W niniejszej pracy dążymy jednak do tego, by unikać metod, które wprowadzają ograniczenia strukturalne rozpoznawanych wyrazów. W związku z tym potrzebna jest metoda automatycznego wyznaczenia wzorców odmiany. Pomysł polega na tym, by wykorzystać linki przychodzące do artykułów. W rozdziale 4.1 pokazano, że linki te zapisywane są w tabeli `Link`. Można więc wykorzystać treść linku (atrybut *text*) do stwierdzenia, które segmenty są odmienne. Powinno to również umożliwić ujednoznacznienie wielu niejednoznacznych segmentów, a także podjęcie decyzji co do tego, czy hasło pisane jest wielką czy małą literą. Kolejną zaletą jest fakt, że hasła, do których nie prowadzą linki, często nie są wyrazami wielosegmentowymi[1], więc możemy je przy okazji odfiltrować. Wada metody polega jednak na tym, że treść linku jest czasami błędna. Powoduje to konieczność zastosowania dość złożonego algorytmu opisanego poniżej.

Tabela 8.1: Statystyka linków przychodzących dla hasła "Czarna dziura".

Treść ($link$)	**Liczba** (N)
czarnej dziury	71
czarna dziura	60
czarnych dziur	46
czarną dziurę	39
czarne dziury	27
Czarna dziura	22
czarną dziurą	16
czarnymi dziurami	7
czarnej dziurze	4
czarnych dziurach	2
metryką	2
czarnym dziurom	1
Czarne dziury	1
Czarna dziura (film)	1
Czarną dziurę	1
osobliwości	1
czarnych dziura	1

8.1. Automatyczne wyznaczanie wzorca odmiany na podstawie linków przychodzących

Celem algorytmu jest wyznaczenie dla danego hasła Wikipedii wzorca odmiany[2], czyli pary $\omega = (tags, decl)$, np. dla hasła "Kwas siarkowy" chcielibyśmy otrzymać rezultat: $tags =$

[1] Obserwacja opiera się przeglądaniu kilkuset losowo wybranych haseł zarówno z linkami przychodzącymi jak i bez nich.
[2] Pojęcie to zdefiniowano w podrozdziale 7.5.1.

$[\{(\texttt{ACA-kwas}, 1)\}, \{(\texttt{CC-siarkowy}, 8)\}]$, $decl =$"$**$". Pierwszym krokiem wyznaczania wzorca odmiany jest sporządzenie listy linków przychodzących dla danego hasła. Zdecydowano, że w przypadku, gdy kilka haseł (wierszy tabeli `WikiEntry`) ma taką samą wartość atrybutu *name*, przetwarzane są one razem. Wynika to z faktu, że niezwykle rzadki jest przypadek, gdy istnieją dwa wyrazy wielosegmentowe o takiej samej formie podstawowej, a odmieniające się w różny sposób[3]. Dla każdego hasła w pobierane są wszystkie linki prowadzące do wierszy `WikiEntry` odpowiadających hasłu w. Następnie tworzona jest ich statystyka $Stat$ – lista par $(link, N)$, gdzie $link$ to treść linku, a N – liczba wystąpień. Statystyka uporządkowana jest malejąco względem N. Przykładowa statystyka linków dla hasła "Czarna dziura" pokazana została w tab. 8.1. Przykład pokazuje, że nie wszystkie linki są poprawne. O ile jednak wyrażenia "metryką" czy "osobliwości" łatwo rozpoznać, to ostatni link "czarnych dziura" zawiera poprawne segmenty i dopiero analiza statystyczna pozwala stwierdzić, że przeważa opcja, w której słowo "dziura" jest odmienne i link ten jest niepoprawny (odmieniono tylko słowo "czarnych").

8.1.1. Wyznaczanie możliwości odmiany pojedynczego linku

Dla każdego z linków w statystyce należy określić możliwe wzorce odmiany, aby następnie wyznaczyć największy podzbiór linków z niesprzecznym wzorcem odmiany. Zastanówmy się najpierw nad pojedynczym segmentem – dany mamy segment hasła s_0 i odpowiadający mu segment linku s. Zakładając, że segment ten jest odmienny, możemy wyznaczyć możliwości odmiany dla danego segmentu przy pomocy funkcji $tag_changes$ zdefiniowanej następująco[4]:

$$tag_changes(s_0, s) := \{(id, f_0, f, g_0, g) : (id, f_0) \in tag_i(s_0) \wedge (id, f) \in tag_i(s) \\ \wedge (f_0, g_0, g) \in changes(id, f)\} \tag{8.1}$$

Przykład. Rozważmy segment hasła $s_0 =$ "czarna" i odpowiadający mu segment linku $s =$ "czarnej". Przy pomocy CLPM uzyskujemy znaczniki słownikowe: $tag_i(s_0) = \{(\texttt{CC-czarny}, \{15, 21\})\}$, $tag_i(s) = \{(\texttt{CC-czarny}, \{16, 17, 20\})\}$. Wyznaczamy teraz zbiór trójek (f_0, g_0, g) zwracanych przez funkcję $changes$ dla elementów $tag_i(s)$:

Numer formy f	**Opis formy** f	**Trójki** $changes(f)$
16	D r.ż. l.p.	$(15, 1, 2)$
17	C r.ż. l.p.	$(15, 1, 3)$
20	Ms r.ż. l.p.	$(15, 1, 6)$

Okazuje się, że dla wszystkich trójek $f_0 = 15$ oraz $(id, f_0) \in tag_i(s_0)$. W takim razie wynikowy zbiór $tag_changes(s_0, s)$ zawiera 3 elementy: $(\texttt{CC-czarny}, 15, 16, 1, 2)$, $(\texttt{CC-czarny}, 15, 17, 1, 3)$ oraz $(\texttt{CC-czarny}, 15, 20, 1, 6)$.

Przeanalizujmy teraz metodę wyznaczania możliwości odmiany całego linku – jej przebieg pokazuje algorytm 5. Na wejściu dane mamy dwie listy segmentów: segmenty hasła $Segments_0$ i segmenty

[3] Np. jest kilka miejscowości "Nowa Biała" posiadających osobne artykuły, ale nazwa ta posiada jeden wzorzec odmiany.
[4] Przez tag_i oznaczamy wariant funkcji tag, który jest niewrażliwy na wielkość liter.

Algorytm 5: Wyznaczanie możliwości odmiany linku.

Wejście : Segmenty hasła $Segments_0$, segmenty linku $Segments$, liczba segmentów n
Wyjście : Słownik D przechowujący możliwości odmiany linku

1 $D \leftarrow$ `dict()`
2 $D[\eta] \leftarrow []$
3 **for** $i = 0$ **to** $n - 1$ **do**
4 $\quad s_0 \leftarrow Segments_0[i]; s \leftarrow Segments[i]$
5 $\quad T \leftarrow tag_changes(s_0, s)$
6 $\quad D_{next} \leftarrow$ `dict()`; $D_i \leftarrow$ `dict()`
7 $\quad$ **foreach** $(g_0, g) : \exists(id, f_0, f, g_0, g) \in T$ **do**
8 $\quad\quad D_i[(g_0, g)] \leftarrow \{(id, f_0, f) : (id, f_0, f, g_0, g) \in T\}$
9 $\quad$ **if** $lower(s) = lower(s_0)$ **then**
10 $\quad\quad$ **foreach** $k \in D_i$ **do** $D_i[k] \leftarrow D_i[k] \cup \{\phi\}$
11 $\quad\quad$ **foreach** $k \in D : k \notin D_i$ **do** $D_i[k] \leftarrow \{\phi\}$
12 $\quad$ **foreach** $k \in D_i$ **do**
13 $\quad\quad$ **if** $k \in D$ **then** $D_{next}[k] \leftarrow D[k] + [D_i[k]]$
14 $\quad\quad$ **else if** $\eta \in D$ **then** $D_{next}[k] \leftarrow D[\eta] + [D_i[k]]$
15 $\quad D \leftarrow D_{next}$

linku $Segments$[5], obie o długości n[6]. Rezultatem działania algorytmu ma być natomiast struktura D, która obrazuje **możliwości odmiany** linku. W algorytmie użyto struktury `dict`, czyli słownika znanego z języka Python[7]. Słownik D będzie przechowywał pary (k, v), gdzie klucz k oznacza albo parę (g_0, g) oznaczającą podstawowy i odmieniony numer formy wyrazu wielosegmentowego, albo specjalną wartość η, oznaczającą, że wartości g_0 i g są nieokreślone. Z kolei wartości v będą listami, w których dla każdego segmentu będzie się znajdować zbiór elementów (id, f_0, f) oznaczających identyfikator i numery form (podstawowy i odmieniony) danego segmentu lub specjalny element ϕ oznaczający, że segment jest nieodmienny.

Początkowo w słowniku D mamy tylko parę $(\eta, [])$. Oznacza to, że wartości g_0 i g są nieokreślone, a lista segmentów jest na razie pusta (linia 2). Teraz dla każdej pary segmentów (s_0, s) wyznaczamy zbiór T (linie 3-5), zawierający piątki zwracane przez funkcję $tag_changes$. Będziemy teraz konstruowali słownik D_i, którego wartości dodane na koniec list będących odpowiednimi wartościami słownika D utworzą nowy słownik D_{next}. W liniach 7-8 następuje grupowanie elementów T według par (g_0, g) – odpowiada to przypadkowi, gdy bieżący segment jest odmienny. Następnie jeżeli s_0 i s są identyczne po skonwertowaniu do małych liter[8] (linia 9), wówczas bieżący segment może być nieodmienny. W związku

[5] Separatory (np. spacje) nie wchodzą w skład tych list.
[6] Jeżeli długości są różne, taki link możemy od razu odrzucić jako niepoprawny.
[7] Jest to tablica asocjacyjna, przechowująca pary klucz - wartość. Wyrażenie $D[k]$ pobiera lub zapisuje wartość dla klucza k słownika D, a $k \in D$ sprawdza, czy klucz k znajduje się w słowniku D.
[8] Algorytm niewrażliwy na wielkość liter daje lepsze wyniki testów.

z tym dodajemy do wszystkich wartości słownika D_i element ϕ (linia 10). Trzeba zrozumieć, że segment nieodmienny pasuje też do wszystkich kluczy słownika D[9], więc należy dodać odpowiednie pary do D_i (linia 11). Teraz trzeba scalić zawartości D i D_i do D_{next}: dla każdego klucza, który jest zarówno w D jak i w D_i mamy zgodność odmiany – wówczas możemy dodać nowy element na koniec listy (linia 13), natomiast jeżeli w D jest klucz η, oznacza to, że lista $D[\eta]$ odpowiadać będzie każdemu kluczowi k i dlatego można ją wykorzystać dodając na jej koniec nowy element i zapisując pod kluczem k[10] (linia 14). Na koniec D zastępujemy przez D_{next} (linia 15).

Przykład 1. Rozważmy link "czarnej dziury" do hasła "czarna dziura". W pierwszym kroku $s_0 =$ "czarna" i $s =$ "czarnej", czyli mamy przypadek opisany w poprzednim przykładzie, w związku z czym w linii 5 wyznaczamy $T = \{(\texttt{CC-czarny}, 15, 16, 1, 2), (\texttt{CC-czarny}, 15, 17, 1, 3), (\texttt{CC-czarny}, 15, 20, 1, 6)\}$. W liniach 7-8 wyznaczamy słownik D_i zawierający odpowiednie pary. Linie 9-11 nie wykonają się (segment jest na pewno odmienny). Przeanalizujmy teraz najważniejszą operację wyznaczania D_{next} w liniach 12-14. Dla każdego klucza okazuje się, że nie ma go jeszcze w D, jednak istnieje klucz η z wartością [] (pusta lista) – zostanie więc on wykorzystany we wszystkich 3 przypadkach:

Klucz k	$D_i[k]$	$D[k]$	$D[\eta]$	$D_{next}[k]$
$(1, 2)$	$\{(\texttt{CC-czarny}, 15, 16)\}$	–	[]	$[\{(\texttt{CC-czarny}, 15, 16)\}]$
$(1, 3)$	$\{(\texttt{CC-czarny}, 15, 17)\}$	–	[]	$[\{(\texttt{CC-czarny}, 15, 17)\}]$
$(1, 6)$	$\{(\texttt{CC-czarny}, 15, 20)\}$	–	[]	$[\{(\texttt{CC-czarny}, 15, 20)\}]$

Po pierwszym kroku okazuje się więc, że pierwszy segment jest odmienny i podobnie jak cały wyraz występuje w dopełniaczu, celowniku bądź miejscowniku liczby pojedynczej. Zauważmy, że w nowym słowniku D nie ma już klucza η. Rozważmy teraz drugą parę segmentów: $s_0 =$ "dziura", $s =$ "dziury". Wyznaczamy zbiór $T = \{(\texttt{ADA-dziura}, 1, 2, 1, 2), (\texttt{ADA-dziura}, 1, 8, 1, 8), (\texttt{ADA-dziura}, 1, 11, 1, 11), (\texttt{ADA-dziura}, 1, 14, 1, 14)\}$. Oznacza to, że możliwe formy to dopełniacz l.p. lub 3 przypadki (M, B, W) l.mn. Wyznaczamy teraz D_i (będą w nim 4 pary), a następnie D_{next}:

Klucz k	$D_i[k]$	$D[k]$	$D_{next}[k]$
$(1, 2)$	$\{(\texttt{ADA-dziura}, 1, 2)\}$	$[\{(\texttt{CC-czarny}, 15, 16)\}]$	$[\{(\texttt{CC-czarny}, 15, 16)\},$ $\{(\texttt{ADA-dziura}, 1, 2)\}]$
$(1, 8)$	$\{(\texttt{ADA-dziura}, 1, 8)\}$	–	–
$(1, 11)$	$\{(\texttt{ADA-dziura}, 1, 11)\}$	–	–
$(1, 14)$	$\{(\texttt{ADA-dziura}, 1, 14)\}$	–	–

Powstały słownik D ma jedną parę: $D[(1, 2)] = [\{(\texttt{CC-czarny}, 15, 16)\}, \{(\texttt{ADA-dziura}, 1, 2)\}]$, więc wyraz wielosegmentowy ma jako podstawową formę mianownik l.p. (1), a w linku występuje w dopełniaczu l.p. (2). Ponadto oba segmenty są odmienne – otrzymujemy ich identyfikatory oraz numery formy podstawowej i odmienionej.

[9] Segment nieodmienny nie ma żadnego wpływu na możliwości odmiany całego hasła.

[10] Klucz η oznacza możliwość, że wszystkie dotychczasowe segmenty były nieodmienne, więc forma może być dowolna.

Przykład 2. Przeanalizujmy teraz jeszcze przypadek, który wydawać się może prostszy: link "czarna dziura". Dla pierwszego segmentu s_0 = "Czarna" i s = "czarna", a $T = \{(\texttt{CC-czarny}, 1, 1, 15, 15), (\texttt{CC-czarny}, 1, 7, 15, 21)\}$. Jeżeli segment jest odmienny, występuje więc w mianowniku lub wołaczu l.p. W linii 9 okazuje się jednak, że segment może być też nieodmienny, stąd w linii 10 dodamy ϕ do wartości dla kluczy $(1, 1)$ i $(1, 7)$, a w linii 11 przypiszemy $\{\phi\}$ do $D_i[\eta]$. Utwórzmy teraz D_{next}:

Klucz k	$D_i[k]$	$D[k]$	$D[\eta]$	$D_{next}[k]$
η	$\{\phi\}$	[]		$[\{\phi\}]$
$(1, 1)$	$\{(\texttt{CC-czarny}, 15, 15), \phi\}$	–	[]	$[\{(\texttt{CC-czarny}, 15, 15), \phi\}]$
$(1, 7)$	$\{(\texttt{CC-czarny}, 15, 21), \phi\}$	–	[]	$[\{(\texttt{CC-czarny}, 15, 21), \phi\}]$

Po przetworzeniu pierwszego segmentu pozostaje więc klucz η, ponieważ nie znaleziono jeszcze segmentu jednoznacznie odmiennego. Dla drugiego segmentu $s_0 = s$ = "dziura", a $T = \{(\texttt{ADA-dziura}, 1, 1, 1, 1)$. Wyznaczanie$D_i$ i D_{next} przebiega następująco:

Klucz k	$D_i[k]$	$D[k]$	$D_{next}[k]$
η	$\{\phi\}$	$[\{\phi\}]$	$[\{\phi\}, \{\phi\}]$
$(1, 1)$	$\{(\texttt{ADA-dziura}, 1, 1), \phi\}$	$[\{(\texttt{CC-czarny}, 15, 15), \phi\}]$	$[\{(\texttt{CC-czarny}, 15, 15), \phi\}, \{(\texttt{ADA-dziura}, 1, 1), \phi\}]$
$(1, 7)$	$\{\phi\}$	$[\{(\texttt{CC-czarny}, 15, 21), \phi\}]$	$[\{(\texttt{CC-czarny}, 15, 21), \phi\}, \{\phi\}]$

Wynikowy słownik D jest nadspodziewanie złożony – zawiera on wszystkie możliwe przypadki:

- $D[\eta] = [\{\phi\}, \{\phi\}]$, czyli cały wyraz może być nieodmienny (może nie być WW).
- $D[(1, 1)] = [\{(\texttt{CC-czarny}, 15, 15), \phi\}, \{(\texttt{ADA-dziura}, 1, 1), \phi\}]$, co oznacza, że jeżeli wyraz występuje w mianowniku, to każdy segment może być odmienny bądź nieodmienny.
- $D[(1, 7)] = [\{(\texttt{CC-czarny}, 15, 21), \phi\}, \{\phi\}]$, czyli jeżeli wyraz występuje w wołaczu, to segment "czarna" jest odmienny lub nieodmienny, a segment "dziura" jest odmienny.

Oprócz opisanych przykładów istnieje kilka bardziej złożonych możliwości, zwłaszcza dla segmentów niejednoznacznych – np. w opisanych przykładach mogliśmy wyznaczyć pojedyncze identyfikatory w CLPM, a nie zawsze tak będzie.

8.1.2. Korekta pierwszej litery hasła

Po wyznaczeniu zbioru linków, dla których słownik D jest niepusty, można wykorzystać je do poprawy pierwszej litery hasła – oryginalnie hasło pisane jest zawsze wielką literą, co wynika z konwencji Wikipedii. Aby stwierdzić, czy należy zmienić tę literę na małą, trzeba wyznaczyć zbiór tych linków, które mogą posłużyć do korekty wielkości litery. W tym celu link musi spełniać przynajmniej jeden z poniższych warunków:

- pierwszy segment jest pisany małą literą,

– link nie występuje na początku zdania,

– znacznik słownikowy pierwszego segmentu zawiera wyłącznie jednostki leksykalne pisane wielką literą.

Wówczas można policzyć dany link jako pewne źródło korekty. Na koniec zliczamy, ile było wystąpień pisanych wielką, a ile małą literą i decyduje głos większości. Podobny algorytm zastosowano też dla pozostałych segmentów hasła – czasami okazuje się, że np. autor hasła błędnie zapisał wszystkie segmenty wielkimi literami, a w linkach pisane są one małymi literami. Korekta wielkości liter ma szczególne znaczenie, ponieważ algorytm wyznaczający możliwości odmiany jest niewrażliwy na wielkość znaków – postanowiono, że jeżeli nie udało się skorygować pierwszej litery hasła (nie było odpowiednich linków), wówczas dane hasło odrzucamy i nie wyznaczamy dla niego wzorca odmiany. Decyzja ta spowodowała poprawę wyników.

8.1.3. Tworzenie zbioru wzorców odmiany

Po wykonaniu dotychczasowych kroków mamy daną statystykę linków $Stat$ wraz ze słownikami możliwości odmiany D dla każdego z linków. Trzeba teraz wyznaczyć wzorce odmiany na podstawie słowników D. Dodatkowo założono, że w wynikowym wzorcu odmiany powinien zostać jednoznacznie określony numer formy podstawowej g_0[11]. Tymczasem istnieją hasła takie jak np. “Małe Ciche”, które mogą być w liczbie pojedynczej ($g_0 = 1$) lub mnogiej ($g_0 = 8$). Co więcej, odmieniona forma “Małym Cichym” sprawia ten sam problem. Dlatego też podjęto decyzję o podzieleniu danych na dwie grupy – jedną dla $g_0 = 1$ i drugą dla $g_0 = 8$. Dla każdej z grup wyznaczymy osobno zbiory wzorców odmiany, a wyniki z obu grup porównamy dopiero na końcu, wybierając wynik z większą **licznością**, czyli łączną liczbą wystąpień linków.

Dla ustalonego g_0 celem jest teraz wyznaczenie zbioru wzorców odmiany Ω. Dla każdego linku tworzymy odpowiadający mu wzorzec $\omega_{link} \in \Omega$ w następujący sposób. Oznaczmy słownik D dla linku $link$ przez D_{link}. Dla każdego linku $link$ wykonujemy teraz poniższą operację:

– Wyznacz **wzorzec odmiany** dla każdej pary (D_{link}, g) takiej, że $(g_0, g) \in D_{link}$ przy pomocy funkcji `create_decl_pattern` (algorytm 6), a następnie scal powstałe wzorce w jeden przy pomocy funkcji `decl_pattern_union`.

Przeanalizujmy pierwszą z tych funkcji. Argument n to liczba segmentów hasła. Funkcja ta konstruuje listę znaczników słownikowych $tags$ i znacznik odmienności segmentów $decl$. Zmienne te są listami o długości n (linia 2). Teraz dla każdego segmentu (linia 3) przypisujemy do i-tego elementu $tags$ wszystkie możliwe pary (id, f_0) dla formy podstawowej tego segmentu (linia 4). Następnie, jeżeli i-ty segment może być nieodmienny (linia 6), do i-tego elementu $decl$ dodajemy element `"-"`, a jeżeli segment ten może być odmienny (czyli zbiór $tags[i]$ jest niepusty), dodajemy element `"*"` (linia 7). W ten sposób powstaje wzorzec odmiany $\omega = (tags, decl)$.

[11] Pozwala to na znaczące uproszczenie dalszego przetwarzania danych.

Algorytm 6: Tworzenie wzorca odmiany dla pojedynczego linku.

```
1  Funkcja create_decl_pattern(D_link, g_0, g, n)
2      tags ← list(n); decl ← list(n)
3      for i = 0 to n − 1 do
4          tags[i] ← {(id, f_0) : ∃(id, f_0, f) ∈ D_link[(g_0, g)][i]}
5          decl[i] ← ∅
6          if φ ∈ D_link[(g_0, g)][i] then decl[i] ← decl[i] ∪ {"-"}
7          if tags[i] ≠ ∅ then decl[i] ← decl[i] ∪ {"∗"}
8      return (tags, decl)
9  Funkcja decl_pattern_union((tags_1, decl_1), (tags_2, decl_2), n)
10     tags ← list(n); decl ← list(n)
11     for i = 0 to n − 1 do
12         tags[i] ← tags_1[i] ∪ tags_2[i]
13         decl[i] ← decl_1[i] ∪ decl_2[i]
14     return (tags, decl)
```

Przykład 1. Przeanalizujmy tworzenie wzorców odmiany dla dwóch przykładowych linków z poprzedniego przykładu. Wystarczy rozpatrzyć wariant dla $g_0 = 1$, bo jest to jedyna opcja dla tych linków. Pierwszy z nich to link "czarnej dziury". Wyznaczyliśmy już słownik D_{link}, który miał tylko jeden klucz $(1, 2)$ i odpowiadającą mu wartość $[\{(\texttt{CC-czarny}, 15, 16)\}, \{(\texttt{ADA-dziura}, 1, 2)\}]$. W takim razie wywołujemy funkcję `create_link_option`$(D_{link}, 1, 2, 2)$. Wykonane w liniach 4-7 iteracje podsumowuje tabela:

i	$D_{link}[(1,2)][i]$	$tags[i]$	$decl[i]$
0	$\{(\texttt{CC-czarny}, 15, 16)\}$	$\{(\texttt{CC-czarny}, 15)\}$	{ "∗" }
1	$\{(\texttt{ADA-dziura}, 1, 2)\}$	$\{(\texttt{ADA-dziura}, 1)\}$	{ "∗" }

W skrócie możemy zapisać wynik jako: $tags = [\{(\texttt{CC-czarny}, 15)\}, \{(\texttt{ADA-dziura}, 1)\}]$, $decl =$ "∗∗".

Przykład 2. Rozważmy teraz drugi link "czarna dziura". W tym przypadku słownik D_{link} był bardziej skomplikowany – klucze to: $(1, 1)$, $(1, 7)$ oraz η (ten element pomijamy). Rozważmy na początku klucz $(1, 1)$, dla którego wywołamy funkcję `create_link_option`$(D_{link}, 1, 1, 2)$:

i	$D_{link}[(1,1)][i]$	$tags[i]$	$decl[i]$
0	$\{(\texttt{CC-czarny}, 15, 15), \phi\}$	$\{(\texttt{CC-czarny}, 15)\}$	{ "∗", "-"}
1	$\{(\texttt{ADA-dziura}, 1, 1), \phi\}$	$\{(\texttt{ADA-dziura}, 1)\}$	{ "∗", "-"}

Wynikowe znaczniki to $tags = [\{(\texttt{CC-czarny}, 15)\}, \{(\texttt{ADA-dziura}, 1)\}]$, natomiast każdy segment może być zarówno odmienny, jak i nieodmienny – będziemy zapisywać to w skrócie jako $decl =$ "??"[12]. Wzorzec ten oznaczmy przez ω_A. Dla klucza $(1, 7)$ wynik jest zbliżony:

[12]Znak zapytania jest używany jako skrócony zapis zbioru { "∗", "-"}.

i	$D_{link}[(1,7)][i]$	$tags[i]$	$decl[i]$
0	$\{(\texttt{CC-czarny}, 15, 21), \phi\}$	$\{(\texttt{CC-czarny}, 15)\}$	$\{$ "$*$", "-"$\}$
1	$\{\phi\}$	$\varnothing$	$\{$ "-"$\}$

Otrzymujemy więc $tags = [\{(\texttt{CC-czarny}, 15)\}, \varnothing]$, $decl =$ "?-". Ten wzorzec oznaczmy przez ω_B. Uzyskane wzorce ω_A i ω_B możemy teraz scalić funkcją `decl_pattern_union`. Jest to prosta operacja polegająca na wyznaczaniu sumy kolejnych par zbiorów będących elementami list $tags$ (linia 12) i $decl$ (linia 13). W omawianym przypadku wynikiem będzie wzorzec identyczny z ω_A. Będziemy to w skrócie zapisywać jako $\omega_A \cup \omega_B = \omega_A$. Podsumujmy teraz wzorce odmiany powstałe w omówionych przykładach:

g_0	$link$	$tags$	$decl$
1	czarnej dziury	$[\{(\texttt{CC-czarny}, 15)\}, \{(\texttt{ADA-dziura}, 1)\}]$	$**$
1	czarna dziura	$[\{(\texttt{CC-czarny}, 15)\}, \{(\texttt{ADA-dziura}, 1)\}]$	??

Wracając teraz do pełnego przykładu dla hasła “Czarna dziura” (tab. 8.1), możemy utworzyć analogiczną tabelę – miałaby ona jednak wiele wierszy z równoważnymi wzorcami odmiany. W rzeczywistości równoważne wzorce powinny zostać scalone celem minimalizacji złożoności. Wynikową listę wzorców przedstawia tab. 8.2. Przez **liczność** wzorca odmiany rozumiemy łączną liczbę wystąpień linków odpowiadających temu wzorcowi.

Tabela 8.2: Wzorce odmiany wygenerowane dla hasła “Czarna dziura”. Wzorcom nadano symbole $\omega_1, \omega_2, \omega_3$ i uporządkowano je według malejącej liczności.

Wzorzec	$tags$	$decl$	**Linki**	**Liczność**
ω_1	$[\{(\texttt{CC-czarny}, 15)\}, \{(\texttt{ADA-dziura}, 1)\}]$	$**$	czarnej dziury, czarnych dziur, czarną dziurę, czarne dziury, czarną dziurą, czarnymi dziurami, czarnej dziurze, czarnych dziurach, czarnym dziurom, Czarne dziury, Czarną dziurę	215
ω_2	$[\{(\texttt{CC-czarny}, 15)\}, \{(\texttt{ADA-dziura}, 1)\}]$	??	czarna dziura, Czarna dziura	82
ω_3	$[\{(\texttt{CC-czarny}, 15)\}, \varnothing]$	$*$-	czarnych dziura	1

8.1.4. Wyznaczanie optymalnego przecięcia wzorców odmiany

Mamy dany zbiór wzorców Ω, a potrzebujemy jednego wzorca dla całego hasła – trzeba więc połączyć istniejące wzorce. Do łączenia wzorców służy funkcja `decl_pattern_intersection` (algorytm 7), wyznaczająca przecięcie wzorców. Funkcja ta jest dość prosta – polega na wyznaczeniu przecięcia

odpowiednich par zbiorów będących elementami list $tags$ (linia 4) i $decl$ (linia 5). Ważny jest jednak warunek w linii 6 – jeżeli okaże się, że wynikowy zbiór $tags[i]$ jest pusty, segment wynikowego wzorca odmiany nie może być już oznaczony jako odmienny (linia 7). Jeżeli okaże się, że w wynikowej liście $decl$ którykolwiek ze zbiorów jest pusty, oznacza to, że wzorce odmiany są sprzeczne ze sobą i operacja nie powiodła się (linia 8). Podsumowując, wywołanie opisywanej funkcji dla pary (ω_1, ω_2) spowoduje zwrócenie albo nowego wzorca będącego ich przecięciem, albo $\varnothing$, gdy przecięcie jest puste. Operację przecięcia będziemy zapisywali w skrócie jako $\omega_1 \cap \omega_2$.

Zauważamy, że problem sprowadza się do znalezienia takiego podzbioru niesprzecznych wzorców $\Psi \subset \Omega$, dla którego łączna liczność będzie największa. Rozwiązanie tego problemu wymaga więc sprawdzenia dla każdego z podzbiorów $\Psi \subset \Omega$, czy przecięcie wszystkich jego elementów jest niepuste. Dla zbiorów spełniających ten warunek możemy wyznaczyć łączną liczność. Pozostaje teraz wybrać podzbiór o największej liczności. Wzorzec będący przecięciem elementów tego zbioru będzie szukanym wzorcem odmiany hasła.

Jedyny problem polega na tym, że istnieją hasła, dla których zbiory Ω mają kilkanaście lub więcej elementów[13] i operacja ta ze względu na wykładniczą złożoność trwałaby zbyt długo. Opracowano więc algorytm optymalizujący wyszukiwanie optymalnego podzbioru – opiera się on na przeszukiwaniu w głąb (DFS) drzewa podzbiorów z odcięciami. Algorytm ten nie stanowi głównego elementu pracy, podamy więc jedynie główne zasady, które są w nim wykorzystywane. Zostaną one przedstawione na przykładzie wzorców dla hasła “Czarna dziura” z tab. 8.2. Przeszukiwanie wykonywane jest w oparciu o drzewo podzbiorów – przedstawiono je na rys. 8.1. Zasadę konstrukcji drzewa dla zbioru wzorców odmiany $\Omega = \{\omega_1, \omega_2, \ldots, \omega_N\}$ można zdefiniować indukcyjnie:

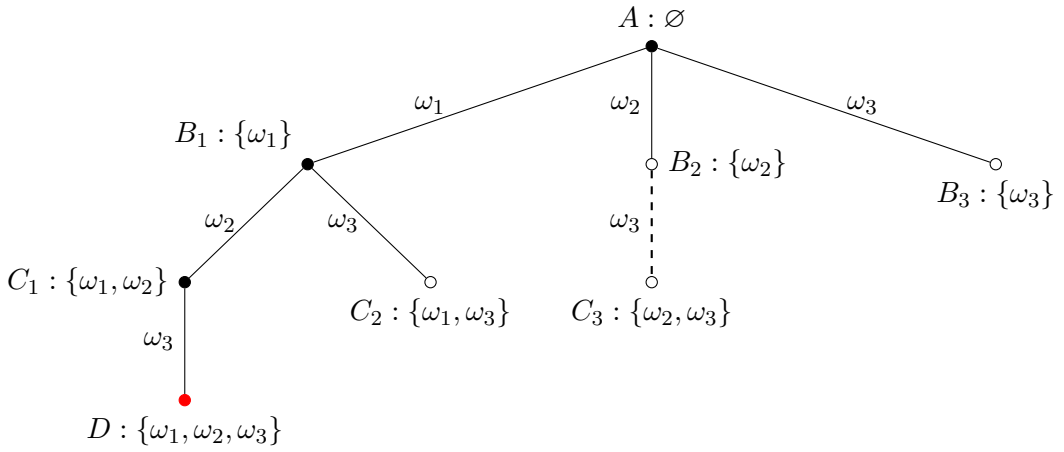
– W korzeniu rozpatrujemy zbiór pusty.

– Jeżeli w węźle n rozpatrujemy podzbiór Ψ, dla którego $m = \max_{\omega_i \in \Psi} i$, to w dzieciach tego węzła będziemy rozpatrywać podzbiory $\Psi \cup \{\omega_{m+1}\}, \Psi \cup \{\omega_{m+2}\}, ..., \Psi \cup \{\omega_N\}$. W przypadku korzenia $m = 0$.

[13] Jest tak dla niektórych haseł posiadających bardzo wiele linków, w tym część błędnych.

Algorytm 7: Wyznaczanie przecięcia wzorców odmiany.

```
1 Funkcja decl_pattern_intersection((tags_1, decl_1), (tags_2, decl_2), n)
2     tags ← list(n); decl ← list(n)
3     for i = 0 to n − 1 do
4         tags[i] ← tags_1[i] ∩ tags_2[i]
5         decl[i] ← decl_1[i] ∩ decl_2[i]
6         if "*" ∈ decl[i] ∧ tags[i] = ∅ then
7             decl[i] ← decl[i] \ { "*" }
8         if decl[i] = ∅ then return ∅
9     return (tags, decl)
```



Rysunek 8.1: Drzewo podzbiorów zbioru wzorców odmiany $\Omega = \{\omega_1, \omega_2, \omega_3\}$. Przy każdym węźle oznaczono rozpatrywany podzbiór $\Psi \subset \Omega$, a przy krawędzi – dodawany element. Wierzchołki czarne oznaczają udaną operację dodania nowego elementu. Wierzchołek czerwony oznacza operację nieudaną i dodanie nowego wyniku do zbioru $Results$. Wierzchołki puste oznaczają miejsca, w których doszło do odcięcia. Krawędzie nie odwiedzone przez algorytm oznaczono linią przerywaną.

W każdym węźle n przechowujemy wzorzec ω_n, będący przecięciem wszystkich wzorców z bieżącego podzbioru. Dla każdej krawędzi drzewa k próbujemy dodać do ω_n nowy wzorzec ω_k, wyznaczając przecięcie $\omega_n \cap \omega_k$. Jeżeli okaże się, że w danym węźle nie można już dodać kolejnego elementu ($\omega_n \cap \omega_k = \varnothing$ lub $\Psi = \Omega$), wówczas bieżący zbiór dodajemy do zbioru wyników $Results$. Dodatkowo stosowane są poniższe reguły redukujące złożoność:

1. Nie dodajemy do $Results$ zbiorów, których nadzbiory już się tam znajdują.
2. Dodając nowy zbiór do $Results$ usuwamy stamtąd jego podzbiory.
3. Warunek odcięcia: jeżeli łączna liczność wzorców rozpatrywanych w bieżącym węźle i wzorców, które zostałyby jeszcze rozpatrzone w jego dzieciach nie przekracza dotychczasowej maksymalnej liczności ze zbioru $Results$, przerywamy przeszukiwanie w bieżącym poddrzewie.

Rozważmy teraz przebieg tej operacji dla przykładowego drzewa. Poniżej wymieniono kolejno rozpatrywane wierzchołki – dla każdego wierzchołka obowiązuje kolejność wzorców według malejącej liczności:

1. A: Wywołanie początkowe.
2. B_1: Rozpatrujemy wzorzec ω_1, który staje się ω_{B_1}.
3. C_1: Przetwarzamy teraz wzorzec ω_2, dla którego udaje się wyznaczyć przecięcie z ω_{B_1}: $\omega_{C_1} = \omega_{B_1} \cap \omega_2 = \omega_1$.
4. D: Następnie przetwarzamy wzorzec ω_3, dla którego przecięcie z ω_{C_1} jest puste (konflikt odmienności drugiego segmentu). Dlatego też do $Results$ dodajemy zbiór $\{\omega_1, \omega_2\}$. Liczność tego zbioru to 297.

5. C_2, B_2, B_3: W tych wierzchołkach dojdzie do odcięcia ze względu na to, że nie jest możliwe uzyskanie wyższej liczności z pominięciem wzorca ω_1 lub ω_2.

Uzyskany zbiór $Results$ zawiera więc jeden wynik $\{\omega_1, \omega_2\}$, który jako jedyny jest oczywiście optymalny. Wzorzec odmiany dla hasła "Czarna dziura" jest więc przecięciem $\omega_1 \cap \omega_2 = \omega_1$ i ma postać następującą: $tags = [\{(\texttt{CC-czarny}, 15)\}, \{(\texttt{ADA-dziura}, 1)\}]$ oraz $decl = \texttt{"**"}$. Jest on zgodny z rzeczywistością (rodzaj żeński, liczba pojedyncza, oba segmenty odmienne). Na koniec przypomnijmy, że wyznaczenie wzorców odmiany przeprowadzaliśmy osobno dla $g_0 = 1$ i $g_0 = 8$. Wybieramy teraz z dwóch wyników ten, który ma wyższą liczność, a w przypadku równej liczności – wynik dla $g_0 = 1$[14]. W rozważanym przypadku nie ma w ogóle linków z $g_0 = 8$, więc problem wyboru nie istnieje.

8.1.5. Proces tworzenia i zapisu wzorców odmiany

Znamy już kroki potrzebne do utworzenia wzorca odmiany dla danego hasła – przeanalizujmy teraz cały proces wraz z zapisywaniem wyników:

1. W pierwszej kolejności tworzona jest statystyka linków przychodzących $Stat$.
2. Następnie dla każdego linku $link$ wyznaczane są jego możliwości odmiany D_{link}.
3. Kolejnym krokiem jest próba korekty pisowni pierwszej litery hasła.
4. Kolejny etap to wyznaczanie zbioru wzorców odmiany Ω i jego niesprzecznego podzbioru Ψ o maksymalnej liczności, osobno dla $g_0 = 1$ i $g_0 = 8$.
5. Wybierany jest lepszy wynik Ψ spośród tych 2 możliwości – przecięcie wzorców z wybranego podzbioru to poszukiwany wzorzec odmiany.
6. Następnie do tabeli `Word` zapisywany jest nowy wiersz odpowiadający wzorcowi odmiany.

Warto podkreślić, że każdy z kroków 1-3 może się nie powieść – wówczas proces przerywamy. Pozostał do omówienia krok ostatni. Tabelę `Word` przedstawiono już wcześniej – przechowuje ona wyrazy wielosegmentowe będące np. etykietami semantycznymi (podrozdział 5.7) lub wynikami rozpoznawania wzorców (podrozdział 7.5.1). W tym wypadku wykorzystano ją do zapisu wyrazów wielosegmentowych, dla których wyznaczyliśmy wzorzec odmiany. Hasło z ustaloną pisownią pierwszej litery zapisujemy jako atrybut *name*, a wartości $tags$ i $decl$ zapisujemy w postaci tekstowej jako atrybuty *tags* i *decl* nowego wiersza tabeli. Specjalny atrybut *is_entry* o wartości `True` oznacza, że dany wyraz został utworzony z hasła Wikipedii. Dodatkowo zapisywane są jeszcze atrybuty oznaczające **jakość** wzorca odmiany. Jak zauważyliśmy, w kroku 4 może dojść do odrzucenia niektórych linków. Jeżeli jest to duży odsetek, istnieje spora szansa na to, że wynikowy wzorzec jest błędny. Dlatego też zdecydowano się wprowadzić dwa dodatkowe atrybuty:

– Atrybut *int_score* oznacza liczbę wystąpień linków odpowiadających wybranemu podzbiorowi wzorców odmiany Ψ, przy czym pomijane są linki, które miały treść identyczną z hasłem[15] –

[14] Empirycznie stwierdzono, że jest to bardziej prawdopodobna opcja.
[15] Z dokładnością do wielkości liter.

nie podnoszą one w żaden sposób jakości wynikowego wzorca, ponieważ nie zachodzi dla nich odmiana.

– Atrybut *score* oznacza iloraz *int_score* i łącznej liczby wystąpień linków, które odpowiadają zbiorowi wszystkich wzorców odmiany Ω z pominięciem linków, które mają treść identyczną z hasłem.

Wartość *score* określa więc, jaka część linków zawierających odmienione hasło została uwzględniona w wybranym wzorcu odmiany. Dla rozpatrywanego w tym rozdziale przykładu *int_score* ma wartość 215, natomiast *score* wynosi $\frac{215}{216} \approx 99.5\%$. Jest to bardzo wysoki wynik – istnieje wiele haseł, dla których jest on dużo niższy. Współczynniki jakości zostaną wykorzystane w algorytmie SM omówionym w kolejnym rozdziale[16].

8.2. Użycie treści artykułu do wyznaczenia wzorca odmiany

Zaprezentowany algorytm wyznaczania wzorców odmiany daje wynik wysokiej jakości, jeżeli tylko gęstość sieci linków jest wystarczająca. Okazuje się, że w przypadku Wikipedii czasem linków brakuje. Można jednak zaobserwować, że często artykuł odpowiadający danemu hasłu zawiera w swoim tekście liczne wystąpienia tego hasła. Nie będą one oczywiście linkami, ponieważ występują już w docelowym artykule. Można takie wystąpienia wykryć skanując cały tekst artykułu, a następnie dodając je do statystyki linków. Dokonując takiej modyfikacji osiągnięto znaczący przyrost jakości wyników. Dodatkowo wskaźnik jakości *int_score* zwiększono o nowe wystąpienia – poprawiło to wyniki dla metody SM (rozdział 9). Nie modyfikowano wskaźnika *score*, ponieważ jakość rozpoznania tych wystąpień jest generalnie niższa niż w przypadku linków – zdarzają się fałszywie pozytywne rozpoznania wystąpień, a poza tym treść niektórych artykułów jest niskiej jakości.

8.3. Testy metody pDM

Metodę DM korzystającą z utworzonych wzorców odmiany nazwano **pDM**. Sposób jej działania i testowania był taki sam, jak dla metody DM – jedyna różnica polega na tym, że wzorce słownikowe tworzone są z uwzględnieniem wzorców odmiany, czyli generowane opcje segmentów ograniczane są przez wartości atrybutów *tags* i *decl*. Przykładowo, dla omawianego w tym rozdziale hasła "Czarna dziura" wzorzec słownikowy wygenerowany dla metody DM ma postać:

```
"Czarna","czarna",WADA-Czarna:1,WCC-czarny:15,wCC-czarny:15 "dziura",wADA-
    ↪ dziura:1
```

W przypadku metody pDM odpowiedni wiersz tabeli `Word` ma atrybuty *name* = "czarna dziura", *tags* = "`CC-czarny:15 ADA-dziura:1`" oraz *decl* = "∗∗". Udało się więc ustalić wielkość pierwszej litery

[16] W algorytmie pDM omawianym w tym rozdziale filtrowanie wzorców po ich wartościach nie przyniosło poprawy rezultatów.

hasła, ujednoznacznić identyfikatory w słowniku oraz stwierdzić, że oba segmenty są odmienne. Wzorzec słownikowy ma więc postać:

```
wCC-czarny:15 wADA-dziura:1
```

Wzorce słownikowe tworzone są tylko dla tych haseł, dla których udało się utworzyć wzorzec odmiany – liczba utworzonych wzorców jest więc dużo niższa niż dla metody DM i wynosi 176506. Tak duże zmniejszenie liczby wzorców musi przełożyć się na zmniejszenie pełności, jednak liczymy na to, że wzrośnie precyzja. Wyniki zaprezentowano w pierwszym wierszu tab. 8.3. Zgodnie z przewidywaniami wartości precyzji wzrosły znacząco (zwłaszcza P_{ext}, por. z tabelą 7.4) przy stosunkowo niewielkim spadku R_{rec}. Wskaźnik F_{rec} jednak nieznacznie zmalał.

Tabela 8.3: Wyniki testów rozpoznawania i ekstrakcji wyrazów wielosegmentowych metodą pDM.

	Test rozpoznawania			**Test ekstrakcji**		
Metoda	P_{rec}	R_{rec}	F_{rec}	P_{ext}	R_{ext}	F_{ext}
pDM (linki + treść artykułu)	**90.12**	38.64	54.09	**86.96**	37.29	52.19
pDM (linki + treść całej Wikipedii)	83.57	40.51	54.57	81.12	39.32	**52.97**
Kombinacja: pDM + DM	80.97	**42.98**	**56.15**	74.19	**39.38**	51.45

Podobnie jak dla DM, tak i tutaj dokonano kilku eksperymentów. Najciekawszy z nich polegał na tym, że wzorzec odmiany wyznaczano nie tylko na podstawie linków przychodzących i wystąpień hasła w artykule, ale także na podstawie wystąpień hasła w całej Wikipedii. Aby umożliwić taką modyfikację, potrzebne było przygotowanie korpusu **WIKI** zawierającego treści artykułów Wikipedii (atrybut *content* tabeli `Article`). Następnie na tym korpusie wykonano rozpoznawanie wszystkich haseł Wikipedii metodą DM, a wyniki zapisywano w prostszej tabeli, zawierającej tylko tekst i kontekst wystąpienia, a także formę podstawową rozpoznanego wyrazu w postaci tekstu. Struktura ta jest więc podobna do tabeli `Link` i była w podobny sposób używana przy tworzeniu wzorców odmiany. Następnie utworzone wzorce posłużyły już do wykonania testu metody pDM. Motywacja do stosowania tego wariantu polegała na tym, że teoretycznie nie wszystkie wystąpienia haseł oznaczane są linkami, więc wykonując pełne rozpoznawanie metodą DM można znaleźć ich więcej[17]. Jest to prawda – podczas tworzenia wzorców odmiany znajdowano około dwukrotnie więcej wystąpień dla każdego hasła, jednak liczba błędów również wzrosła, a ponadto przygotowanie wzorców trwało kilkakrotnie dłużej niż dla zwykłej metody pDM. Wyniki pokazano w 2. wierszu tabeli z wynikami. Na koniec przedstawiono również wynik metody będącej złożeniem pDM i DM (wersja podstawowa – wiersz 1 tab. 7.4) – wyrazy, które nie zostaną rozpoznane przez pDM, rozpoznawane są przy użyciu DM. Mimo że metoda ta osiąga najwyższą dotychczas wartość F_{rec} wynoszącą ponad 56%, to jednak ma dużo niższą precyzję niż pDM.

[17]Można więc powiedzieć, że wariant ten polega na *wikifikacji* treści całej Wikipedii. W przyszłości można by przetestować ten wariant wykorzystując algorytm rankingu przedstawiony w pracy [53].

Podsumowując, metody rozpoznawania wzorców słownikowych uzyskują wysoką precyzję (ponad 90% dla pDM), jednak rozpoznają tylko około 4 na 10 wyrazów wielosegmentowych w tekście[18]. Powoduje to konieczność opracowania dodatkowej metody na rozpoznawanie nowych wyrazów, nie będących hasłami Wikipedii.

[18] Warto zauważyć, że obserwacja ta pokrywa się w dużym stopniu z wynikami NER w pracy [68].

9. Metoda SM

Ekstrakcja WW w oparciu o wzorce syntaktyczne

Dotychczas opisane metody ekstrakcji wyrazów wielosegmentowych, DM i pDM, dokonywały jedynie rozpoznawania wyrazów będących hasłami Wikipedii. Dla podniesienia współczynnika *recall* tych metod konieczne jest wprowadzenie pewnych reguł lub wzorców, które mogłyby posłużyć do ekstrakcji nowych wyrazów. Wzorce takie mogą być definiowane ręcznie – pokazano to na przykładzie heurystyki na rys. 7.6. Podobne podejście zastosowano też np. w pracach [5, 109]. Narzędzia Spejd [8], SProUT [66] i MWEToolkit [78] funkcjonują też w oparciu o zdefiniowane reguły i wzorce. Wydaje się więc, że ciężko osiągnąć dobre wyniki bez ręcznego definiowania syntaktycznych reguł i wzorców lub dostępu do dużej ilości danych treningowych dla klasyfikatorów i tagerów.

Okazuje się jednak, że wiele można osiągnąć wykorzystując opisaną w poprzednim podrozdziale metodę automatycznego wyznaczania wzorców odmiany haseł Wikipedii – skoro dla danego hasła znamy wzorzec odmiany, można wykorzystać jego budowę do znajdowania w tekście wyrazów o podobnej strukturze. Przykładowo wyrazy oznaczające związki chemiczne mają bardzo często podobną budowę – np. dla wyrazów wielosegmentowych "tlenek węgla", "siarczan miedzi", "wodorotlenek sodu" pierwszy segment to odmienny rzeczownik r. męskiego, a drugi – nieodmienny rzeczownik w dopełniaczu. W takim razie jeżeli jedno z tych haseł wystąpiło w Wikipedii i udało się określić dla niego wzorzec odmiany, można próbować odnaleźć w tekście (korpusie) inne wyrazy o takim samym wzorcu odmiany. Główny problem polega na tym, że przetwarzając korpus nie wiemy, jak odmienia się napotkany segment. Możemy jednak sprawdzić, do jakich wzorców odmiany pasuje dane wyrażenie rozpoznane w tekście.

Dodatkowo można pójść o krok dalej i zastanowić się nad strukturą **kontekstu**, w którym występuje wyraz wielosegmentowy[1]. Przykładowo, wymienione związki chemiczne występują często w podobnych wyrażeniach, np. "... zawartość *tlenku węgla* w ...", "... reakcja *siarczanu miedzi* z ...", "... nadmiar *wodorotlenku sodu* w ...". We wszystkich trzech przypadkach wyraz występuje w dopełniaczu, poprzedzony jest rzeczownikiem w mianowniku, a po nim występuje przyimek. Wyszukując wyrazy występujące w podobnym kontekście można więc również znaleźć nowe wyrazy wielosegmentowe.

Przedstawione cechy są stosunkowo proste do wyekstrahowania w sposób automatyczny z haseł Wikipedii – wystarczy odpowiednio rozbudować opracowany już algorytm rozpoznawania wzorca odmiany haseł, tak aby dodatkowo zapisywał dane na temat kontekstu wystąpień danego hasła. Następnie

[1] Zauważono to też np. w pracy [20].

trzeba utworzyć wzorce syntaktyczne opisujące strukturę hasła i kontekstów wystąpień. Na koniec można wykorzystać przedstawiony w podrozdziale 7.3 automat do rozpoznawania tych wzorców w tekście.

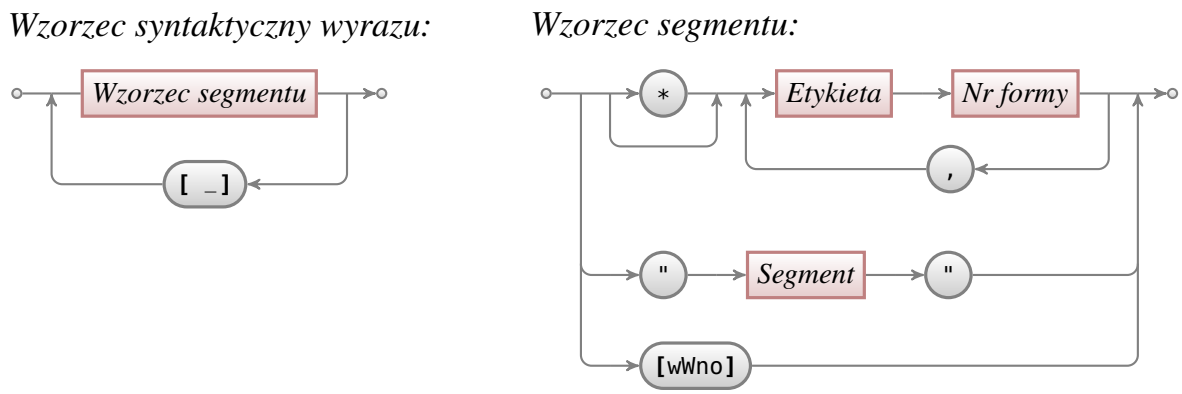
Można zauważyć podobieństwo opisanej koncepcji do działania tagera trenowanego na pewnym zestawie cech syntaktycznych. W naszym przypadku będziemy jednak szukali wyrazów o wartościach cech identycznych z tymi dla haseł Wikipedii. Tagery oparte na metodach statystycznych potrafią z kolei podejmować decyzję dla wartości cech jedynie zbliżonych do przykładowych, jednak potrzebują do tego korpusu treningowego. W przyszłości, gdy korpusy takie będą dostępne, można rozważyć stosowanie takich rozwiązań.

9.1. Wzorce syntaktyczne wyrazów

Pierwszym krokiem do realizacji powyższej koncepcji jest wyekstrahowanie **wzorców syntaktycznych** opisujących strukturę składniową haseł Wikipedii – w pierwszej kolejności opisane zostaną wzorce wyrazów, a dopiero w następnej kolejności zastanowimy się, w jaki sposób zapisać dane o kontekstach wystąpień. Punktem wyjścia jest wzorzec odmiany i utworzony z niego wzorzec słownikowy. Jako przykład rozważmy wyraz “tlenek magnezu”. Załóżmy, że w sposób opisany w poprzednim podrozdziale ustalono jego wzorzec odmiany: $tags = [\{(\texttt{ACA-tlenek}, 1)\}, \{(\texttt{ACA-magnez}, 2)\}]$, $decl =$"$*$-". Spróbujmy teraz zapisać te informacje w postaci jednej listy z pominięciem form tekstowych segmentów: $[(\texttt{"}*\texttt{"}, \texttt{ACA}, 1), (\texttt{"-"}, \texttt{ACA}, 2)]$. Można pominąć trzecią literę etykiety[2] i zapisać całość w postaci tekstu: $*$AC1 -AC2. Jeżeli teraz zmienimy litery etykiet na małe by zilustrować fakt, że segmenty są pisane małymi literami oraz pominiemy znak “-”, otrzymamy postać wynikową: $*$ac1 ac2. Jest to właśnie wzorzec syntaktyczny wyrazu wielosegmentowego. Jak widać, format został zoptymalizowany tak, aby uzyskać zwięzłość zapisu, a przy tym nie stracić czytelności dla człowieka.

Strukturę wzorca syntaktycznego wyrazu wielosegmentowego przedstawiono na rys. 9.1. Podobnie jak w przypadku wzorców słownikowych, wzorzec wyrazu składa się z wzorców segmentu oddzielonych

[2] Oznacza ona jedynie informację, że wyraz zmienia swoją formę tekstową podczas deklinacji.



Rysunek 9.1: Struktura wzorca syntaktycznego wyrazu wielosegmentowego. Wyrażenie [abc] oznacza, że należy wybrać jeden ze znaków a, b, c.

znakiem spacji " "(jeżeli między segmentami występuje spacja) lub znakiem podkreślenia "_" (gdy spacji nie ma). Dla każdego segmentu stosujemy z kolei następujące reguły:

1. Jeżeli segment jest odmienny, wzorzec segmentu zaczynamy od gwiazdki ("*"). Następnie umieszczamy oddzielone przecinkami pary (*etykieta*, *nr_formy*), gdzie *etykieta* dobierana jest następująco:
   - Dla rzeczownika pospolitego są to dwie pierwsze litery etykiety fleksyjnej.
   - Dla rzeczownika będącego nazwą (którego forma podstawowa pisana jest wielką literą) jest to litera "X", po której występuje druga litera etykiety fleksyjnej.[3]
   - Dla przymiotnika i liczebnika wzorcem segmentu jest etykieta fleksyjna.

   Etykieta powinna zostać zapisana małymi literami, jeżeli segment pisany jest małą literą.
2. Jeżeli segment jest słowem nieodmiennym w, dla którego znacznik $tag(w)$ jest niepusty i oznacza rzeczownik, przymiotnik lub liczebnik, wówczas stosujemy te same reguły, co dla segmentu odmiennego z pominięciem gwiazdki "*" na początku.
3. Jeżeli segment jest przyimkiem lub znakiem interpunkcyjnym, wówczas wzorcem segmentu jest sam segment, zapisany w cudzysłowie.[4]
4. Gdy segment jest innym słowem nieodmiennym, wzorcem segmentu jest litera "W", gdy segment pisany jest wielką literą lub "w", gdy pisany jest mała literą.
5. Gdy segment jest liczbą, wzorzec ma postać "n".
6. Dla innych segmentów stosowana jest litera "o".

Kilka przykładowych wzorców syntaktycznych utworzonych na podstawie wzorców odmiany podano poniżej.

Wyraz	**Wzorzec syntaktyczny**
czarna dziura	`∗cc15 ∗ad1`
Droga pod Reglami	`∗AD1 "pod" AC12`
Jan III Sobieski	`∗XA1 o ∗XA1`
bieg na 2000 metrów z przeszkodami	`∗ac1 "na" n aa9,aa11,ac9,ae9 "z" ad12`
choroba Leśniowskiego-Crohna	`∗ad1 XA2,XA4_"-"_W`

W przypadku dwóch ostatnich wzorców widzimy pewne nadmiarowe opcje: segment "metrów" został m.in. rozpoznany jako forma wyrazu "metrum", a z kolei dla słowa "Leśniowskiego" jednym z możliwych przypadków jest biernik. Niestety tego rodzaju niejednoznaczności są niemożliwe do wyeliminowania bez wprowadzenia ręcznie zdefiniowanych reguł gramatycznych. Mimo to widać, że wzorce są dość zwięzłe i dobrze odzwierciedlają strukturę wyrazów.

[3] Pozwala to na odróżnienie np. imion i nazwisk od nazw zawierających wyrazy pospolite, co poprawia skuteczność ekstrakcji.

[4] Z testów wynika, że przyimki i znaki interpunkcyjne często definiują pewne charakterystyczne klasy wyrazów wielosegmentowych i warto je zapisywać dosłownie. Ponadto liczba możliwości jest tutaj nieduża.

9.2. Statystyka wystąpień wzorca syntaktycznego

W podrozdziale 7.5.1 przedstawiono koncepcję oceny wyników ekstrakcji wyrazów wielosegmentowych – problem w metodzie DM polegał na tym, że każdy z rozpoznawanych wzorców był wart 1 punkt. Jest to uzasadnione dla wzorców słownikowych, gdzie rozpoznanie wzorca oznacza wystąpienie konkretnego wyrazu. W przypadku wzorców syntaktycznych często dojdzie jednak do sytuacji, w której dla jednego wyrażenia rozpoznanych zostanie wiele wzorców, a pewność każdego z dopasowań jest dużo niższa niż dla wzorców słownikowych[5]. W takim przypadku dobrze byłoby, gdyby każdy wzorzec syntaktyczny posiadał pewną liczbę oznaczającą jego ważność. Naturalnym wyborem wydaje się łączna **liczba wystąpień** wzorca – będzie to suma liczby wystąpień wyrazów (a konkretnie: liczności przecięcia wzorców odmiany, patrz podrozdział 8.1.4), z których utworzono dany wzorzec syntaktyczny. Przykładowo, dla hasła "Czarna dziura" dającego wzorzec syntaktyczny `*cc15 *ad1` liczność wynosiła 297. Jeżeli ten sam wzorzec powstałby np. dla hasła "Czarna wdowa" z licznością 303, to liczba wystąpień tego wzorca syntaktycznego wynosiłaby 600.

Okazało się jednak, że zauważalną poprawę wyników[6] można uzyskać dodając do liczby wystąpień informację o numerze formy, tworząc tym samym **statystykę wystąpień** wzorca, obrazującą ile razy dany wzorzec występuje w formie o danym numerze. Dzięki temu, jeżeli w wynikach ekstrakcji wyrazów wystąpi niejednoznaczność, często występujące (czyli bardziej prawdopodobne) formy będą wyżej punktowane.

9.2.1. Tworzenie statystyki wystąpień

Analizując ponownie metodę wyznaczania wzorca odmiany możemy dostrzec, że słowniki możliwości odmiany D_{link} wyznaczane dla każdego z linków zawierają jako klucze pary (g_0, g). Numery form g z tych par powinny trafić do statystyki wystąpień. Niestety nie można wyznaczyć ich wprost na podstawie słowników D_{link}, ponieważ chcielibyśmy uwzględnić tylko te składowe wzorce, które trafią później do wyznaczonego wzorca odmiany[7]. W takim razie trzeba rozszerzyć dane wzorca odmiany tworzonego przy pomocy funkcji `create_decl_pattern` (algorytm 6). Aby nie komplikować istniejącego algorytmu, dodatkowe dane będziemy przechowywać w słowniku Γ: dane dla wzorca ω oznaczymy przez $\Gamma[\omega]$. Początkową wartość $\Gamma[\omega]$ dla każdego wzorca $\omega = \texttt{create_decl_pattern}(D_{link}, g)$ definiujemy następująco:

$$\Gamma[\omega] = \{(link, \omega, g)\} \tag{9.1}$$

Podczas scalania wzorców przy pomocy funkcji `decl_pattern_union` nie może dojść do ograniczenia wartości $decl$, możemy więc po prostu dokonać sumy odpowiednich wartości ze słownika Γ:

[5] Przypadkowa zgodność kategorii gramatycznych jest znacznie bardziej prawdopodobna niż zgodność całego wyrazu.
[6] Testy wykazały wzrost wartości F_1 o 1%.
[7] Niektóre mogą zostać odrzucone podczas wyznaczania operacji przecięcia wzorców.

$$\Gamma[\omega_1 \cup \omega_2] = \Gamma[\omega_1] \cup \Gamma[\omega_2] \tag{9.2}$$

Dodać należy, że w przypadku scalania wzorców dla różnych linków (patrz podrozdział 8.1.3 i tab. 8.2) należy również wykonać powyższą operację. Pozostała teraz do rozważenia operacja przecięcia wzorców `decl_pattern_intersection`. W tym przypadku sytuacja jest trudniejsza, ponieważ niektóre składowe wzorce ω mogą zostać odrzucone – należy je więc wyeliminować:

$$\Gamma[\omega_1 \cap \omega_2] = \{(link, \omega, g) \in \Gamma[\omega_1] \cup \Gamma[\omega_2] : \omega \cap (\omega_1 \cap \omega_2) \neq \varnothing\} \tag{9.3}$$

Jako przypadek uzasadniający potrzebę odrzucania niektórych wzorców rozważmy hasło “Muzeum edukacyjne”. Załóżmy, że linki przychodzące to: l_1 ="muzeum edukacyjnym", l_2 ="muzeów edukacyjnych", l_3 ="Muzea edukacyjne". Liczności tych linków wynoszą odpowiednio 3, 2, 1. Poniżej przedstawiono poszczególne składowe wzorce segmentów przed ich scaleniem dla każdego linku. Rozważania ograniczono do przypadku $g_0 = 1$.

Wzorzec	g_0	g	$link$	$tags$	$decl$
ω_{1A}	1	10	l_1	$[\varnothing, \{(\texttt{CC-edukacyjny}, 22)\}]$	`-*`
ω_{1B}	1	5	l_1	$[\{(\texttt{AEA-muzeum}, 1)\}, \{(\texttt{CC-edukacyjny}, 22)\}]$	`?*`
ω_{1C}	1	6	l_1	$[\{(\texttt{AEA-muzeum}, 1)\}, \{(\texttt{CC-edukacyjny}, 22)\}]$	`?*`
ω_2	1	9	l_2	$[\{(\texttt{AEA-muzeum}, 1)\}, \{(\texttt{CC-edukacyjny}, 22)\}]$	`**`
ω_{3A}	1	8	l_3	$[\{(\texttt{AEA-muzeum}, 1)\}, \{(\texttt{CC-edukacyjny}, 22)\}]$	`*?`
ω_{3B}	1	11	l_3	$[\{(\texttt{AEA-muzeum}, 1)\}, \{(\texttt{CC-edukacyjny}, 22)\}]$	`*?`
ω_{3C}	1	14	l_3	$[\{(\texttt{AEA-muzeum}, 1)\}, \{(\texttt{CC-edukacyjny}, 22)\}]$	`*?`

Uwagę zwraca wzorzec ω_{1A}, odpowiadający wyłącznie przypadkowi, gdyby segment “muzeum” był nieodmienny. Poniżej przedstawiono stan po scaleniu wzorców dla każdego linku. W tym przypadku nie ma równoważnych wzorców dla różnych linków, więc żadne wzorce nie będą już później scalane.

Wzorzec	$tags$	$decl$	**Wartość** Γ
ω_1	$[\{(\texttt{AEA-muzeum}, 1)\}, \{(\texttt{CC-edukacyjny}, 22)\}]$	`?*`	$\{(l_1, \omega_{1A}, 10), (l_1, \omega_{1B}, 5), (l_1, \omega_{1C}, 6)\}$
ω_2	$[\{(\texttt{AEA-muzeum}, 1)\}, \{(\texttt{CC-edukacyjny}, 22)\}]$	`**`	$\{(l_2, \omega_2, 9)\}$
ω_3	$[\{(\texttt{AEA-muzeum}, 1)\}, \{(\texttt{CC-edukacyjny}, 22)\}]$	`*?`	$\{(l_3, \omega_{3A}, 8), (l_3, \omega_{3B}, 11), (l_3, \omega_{3C}, 14)\}$

Teraz wyznaczamy optymalne przecięcie wzorców ω_n, które wyraża się wzorem $\omega_1 \cap \omega_2 \cap \omega_3$. W wynikowym wzorcu wszystkie segmenty są jednoznacznie odmienne[8], więc okazuje się, że trzeba wykluczyć ω_{A1}, czyli $\Gamma[\omega_n] = \{(l_1, \omega_{1B}, 5), (l_1, \omega_{1C}, 6), (l_2, \omega_2, 9), (l_3, \omega_{3A}, 8), (l_3, \omega_{3B}, 11), (l_3, \omega_{3C}, 14)\}$. Oznacza to, że ujednoznaczniono formę dla linku "muzeum edukacyjnym" do liczby pojedynczej. Możemy teraz wyznaczyć docelową statystykę wystąpień – ma ona postać zbioru par (G, N_G), gdzie G to zbiór numerów form wyrazu, a N_G – łączna liczność linków, którym w wynikowym zbiorze $\Gamma[\omega_n]$ odpowiada zbiór form G. Dla rozważanego przykładu statystyka ma postać[9]:

Zbiór form G	**Linki**	**Liczność** N_G
5, 6	muzeum edukacyjnym	3
9	muzeów edukacyjnych	2
8, 11, 14	Muzea edukacyjne	1

Zastanówmy się teraz, jak połączyć statystyki wystąpień dla kilku wyrazów. W omawianym przypadku docelowy wzorzec to `*ae1 *cc22`. Taki sam wzorzec może wystąpić np. dla hasła "Badanie cytologiczne", dla którego statystyka wystąpień będzie miała postać:

Zbiór form G	**Linki**	**Liczność** N_G
1, 4, 7	badanie cytologiczne, Badanie cytologiczne	4
2	badania cytologicznego	2
6	badaniu cytologicznym	1
9	badań cytologicznych	1

Wówczas wynikowa statystyka powinna wyglądać następująco:

Tabela 9.1: Wynikowa statystyka wystąpień wzorca syntaktycznego wyrazu.

Zbiór form G	**Linki**	**Liczność** N_G
1, 4, 7	badanie cytologiczne, Badanie cytologiczne	4
5, 6	muzeum edukacyjnym	3
9	muzeów edukacyjnych, badań cytologicznych	3
2	badania cytologicznego	2
6	badaniu cytologicznym	1
8, 11, 14	Muzea edukacyjne	1

Nastąpiło połączenie wierszy dla formy nr 9, jednak nie można połączyć wiersza dla zbioru form $\{5, 6\}$ z wierszem dla jednoelementowego zbioru form $\{6\}$, ponieważ zbiory te są różne. Nie jest możliwe również rozbicie zbiorów na pojedyncze formy – stracilibyśmy wówczas możliwość policzenia wystąpień linków, bo każdy z nich mógłby wystąpić wielokrotnie.

[8] $decl =$ "$**$"

[9] Linki nie wchodzą w skład statystyki, są pokazane jedynie dla zwiększenia czytelności.

9.3. Wzorce syntaktyczne kontekstu

Linki przychodzące służące do wyznaczania wzorca odmiany można również wykorzystać do utworzenia **wzorców syntaktycznych kontekstu**. Pierwszym krokiem jest pobranie lewego i prawego kontekstu dla każdego linku[10]. Kilka przykładowych linków wraz z kontekstem dla hasła "Czarna dziura" przedstawiono poniżej. Konteksty ograniczane są do pojedynczych zdań.

Lewy kontekst	Treść linku	Prawy kontekst
... masa jej centralnej	*czarnej dziury*	jest wyjątkowo mała ...
... zderzeń gwiazd neutronowych,	*czarnych dziur*	lub wybuchów supernowych ...
... układy podwójne gwiazd z	*czarnymi dziurami*	i gwiazdami neutronowymi ...
Ciało takie nazywamy	*czarną dziurą*	.
.../dev/null jest synonimem	*czarnej dziury*	i komputerowego śmietnika ...

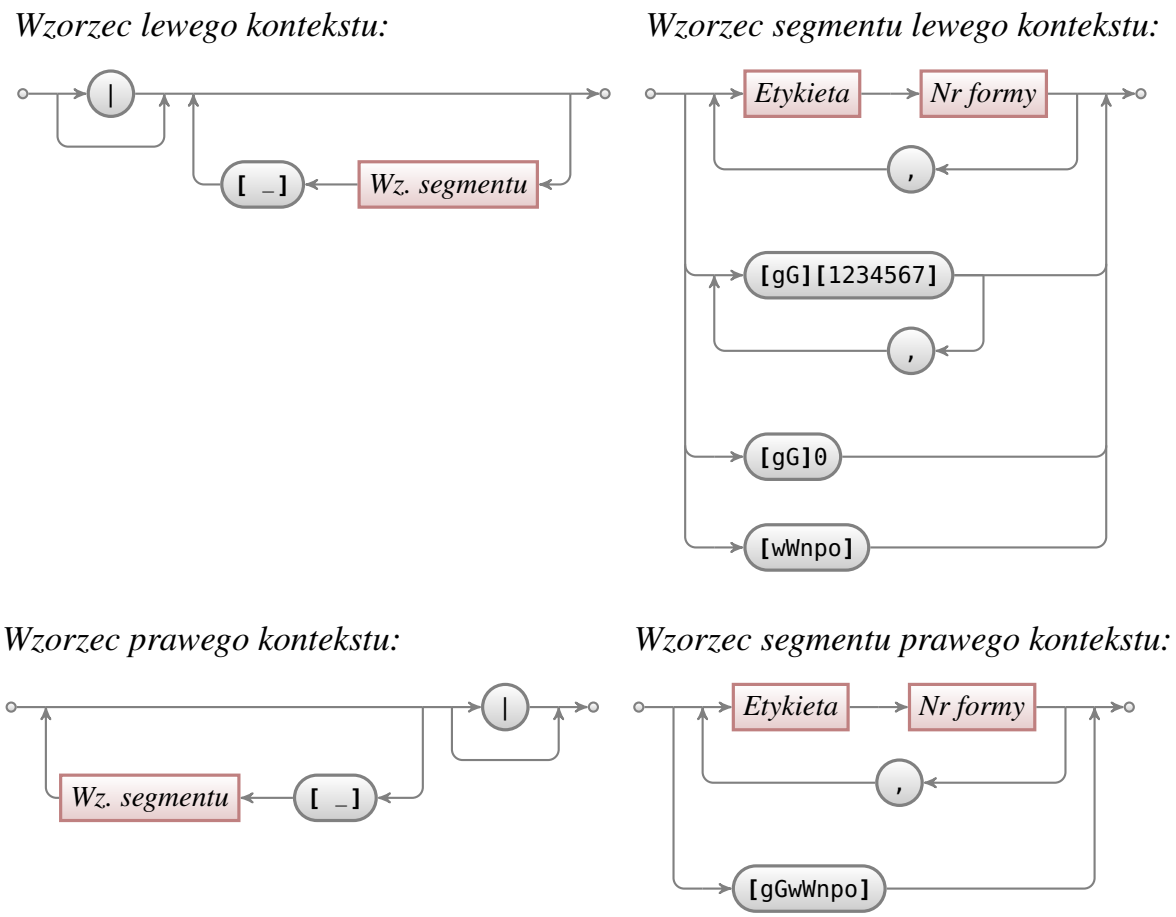
Należy ustalić liczbę segmentów lewego i prawego kontekstu, które służą do wyznaczenia wzorców kontekstu. Załóżmy, że interesuje nas 1 segment dla każdego z kontekstów. Dla każdego linku wyznaczymy teraz listę wzorców kontekstu. Dla przykładu rozważymy poniższe wystąpienia linku "czarnej dziury" (dokonano już wyodrębnienia jednosegmentowych kontekstów):

Lewy kontekst	Spacja?	Treść linku	Spacja?	Prawy kontekst
centralnej	TAK	czarnej dziury	NIE	.
masywnej	TAK	czarnej dziury	NIE	.
lub	TAK	czarnej dziury	NIE	)
Oddziaływania	TAK	czarnej dziury	TAK	tworzą
do	TAK	czarnej dziury	TAK	z

Ogólna struktura wzorca lewego i prawego segmentu podana jest na rys. 9.2. Omówmy na początek wzorzec lewego kontekstu. Uwagę zwraca znak "|" – używany jest on wówczas, gdy nie ma odpowiedniej liczby segmentów lewego kontekstu, ponieważ link występuje na początku zdania. Pozostałe elementy to wzorce kolejnych segmentów, zapisywane podobnie, jak dla wzorca syntaktycznego wyrazu – jedyna różnica polega na tym, że na końcu występuje separator (" " lub "_"). W przypadku kontekstów o długości 1 wzorzec lewego kontekstu może zawierać sam znak "|" lub pojedynczy segment i separator. Dla prawego kontekstu sytuacja przedstawia się analogicznie – mamy do czynienia z odbiciem lustrzanym struktury lewego kontekstu.

Skupmy się teraz na wzorcach segmentów lewego i prawego kontekstu. Obowiązują tu podobne reguły jak dla wzorca wyrazu – oczywiście z pominięciem wzorców segmentu odmiennego. Ponadto podczas eksperymentów okazało się, że niektóre informacje są zbyt szczegółowe i przez to przestrzeń wzorców pokryta jest zbyt rzadko. Podjęto więc próby mające na celu ograniczenie szczegółowości wzorców, pozostawiając jednocześnie najważniejsze ich cechy. Dla lewego kontekstu zmiany są następujące:

[10] Są to atrybuty *left_context* i *right_context* tabeli `Link`.



Rysunek 9.2: Struktura wzorca syntaktycznego lewego i prawego kontekstu. Wyrażenie `[abc]` oznacza, że należy wybrać jeden ze znaków `a`, `b`, `c`.

1. Znaków interpunkcyjnych jest zbyt wiele, by zapisywać je dosłownie, więc stosujemy dla nich wzorzec segmentu "p".
2. Dla przyimka zapisujemy literę "g" (gdy jest pisany małą literą) lub "G" (gdy piszemy go wielką literą), a następnie oczekiwany numer przypadka rzeczownika. Gdy jest kilka przypadków, piszemy kilka takich par, oddzielając je przecinkami, np. dla przyimka "w" utworzymy wzorzec segmentu `g4,g6`, ponieważ po nim może wystąpić rzeczownik w bierniku lub miejscowniku.
3. Dla innych nieodmiennych części mowy z niepustym znacznikiem słownikowym stosujemy wzorzec "G0".

W przypadku prawego kontekstu również stosujemy regułę 1, a ponieważ tutaj przypadek rzeczownika dla przyimków nie ma znaczenia, dlatego wszystkim nieodmiennym częściom mowy z niepustym znacznikiem słownikowym przypisujemy wzorzec "g" lub "G" (w zależności od pisowni).

Znając już reguły tworzenia wzorców kontekstu, utwórzmy odpowiednie wzorce dla przykładowych linków – pokazano je w tab. 9.2. Każdą parę wzorców kontekstu można połączyć z wzorcem wyrazu, tworząc pełny **wzorzec syntaktyczny z kontekstem**, np. `cc16,cc17,cc20 *cc15 *ad1_p`[11]. Przypomnijmy

[11] Wzorzec syntaktyczny wyrazu dla hasła "czarna dziura" to `*cc15 *ad1`.

Tabela 9.2: Wzorce kontekstu dla linku “czarnej dziury”.

L. kontekst	Spacja?	Wz. l. kontekstu	Spacja?	P. kontekst	Wz. p. kontekstu
centralnej	TAK	`"cc16,cc17,cc20 "`	NIE	.	`"_p"`
masywnej	TAK	`"cc16,cc17,cc20 "`	NIE	.	`"_p"`
lub	TAK	`"g0 "`	NIE	)	`"_p"`
Oddziaływania	TAK	`"AE2,AE8,AE11,AE14 "`	TAK	tworzą	`" b7"`
do	TAK	`"g2 "`	TAK	z	`" g"`

teraz statystykę wystąpień wprowadzoną w poprzednim podrozdziale, tworzoną dla wzorca wyrazu – możemy również utworzyć taką statystykę dla każdego wzorca z kontekstem. To pozwoli np. uwzględnić fakt, że powyższy wzorzec odpowiada formie dopełniacza liczby pojedynczej wyrazu. Zamiast liczności linków będziemy teraz zapisywać liczbę wystąpień danego wzorca z kontekstem – przykładowo dla wzorca z kontekstem `cc16,cc17,cc20 *cc15 *ad1_p` utworzymy poniższą statystykę, uwzględniając jego dwa wystąpienia z tab. 9.2:

Zbiór form G	**Linki**	**Liczba wystąpień**
2	czarnej dziury	2

Po przetworzeniu hasła “Czarna dziura” do końca oraz kolejnych haseł może ona wyglądać następująco[12]:

Zbiór form G	**Przykłady linków**	**Liczność**
2	Drewnianej Łyżki, Piątej kolumny, Platynowej Płyty, Platynowej płyty, białej rasy, całej nuty, czarnej dziury, czarnej skrzynki, drewnianej łyżki, gorącej czekolady, pierwszej damy, piątej kolumny, (. . .)	43
2, 3, 6	Chrześcijańskiej Demokracji, Chrześcijańskiej demokracji, Intensywnej Terapii, Krwawej Niedzieli, Krwawej niedzieli, Nowej Fali, Nowej fali, Pierwszej Pomocy, Pomarańczowej Rewolucji, (. . .)	29
3, 6	brudnej wojnie, czarnej liście, podwójnej helisie, wolnej burcie	5
1	Betonowa granica	1
9	czarnych dziur	1

Ostatnie dwa wiersze powstały z nietypowych konstrukcji składniowych, np. “dzięki ścieżce dydaktycznej *Betonowa granica*.” i mogą spowodować zmniejszenie poprawności wyników ekstrakcji wyrazów. Dlatego też istotne jest tworzenie statystyk liczbowych, które zmniejszają ich wpływ na wynik.

9.4. Zapis wzorców syntaktycznych

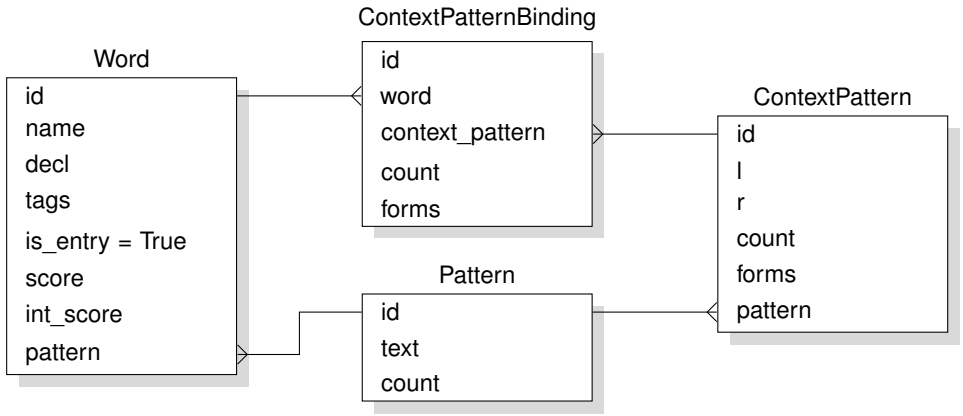
Docelową strukturę bazy danych przedstawiono na rys. 9.3. Podstawową tabelą jest `Pattern` – każdy jej wiersz zawiera pojedynczy wzorzec syntaktyczny wyrazu. Z wyrazami wielosegmentowymi (`Word`),

[12]Niektóre linki pisane są wielką literą, lecz dotyczą one wyrazów, których pisownia została poprawiona do małej litery (patrz podrozdział 8.1.2).

dla których udało się utworzyć wzorce odmiany[13] łączy ją relacja jeden-do-wielu, ponieważ taki sam wzorzec można utworzyć z wielu różnych wyrazów. Aby można było zapisać tę relację, do tabeli `Word` dodano atrybut *pattern*. Atrybut *text* wzorca przechowuje jego treść, a *count* – łączną liczbę wystąpień dla wszystkich wyrazów. Pozwala ona na łatwą selekcję wzorców mających np. więcej niż 10 wystąpień.

Kolejna istotna tabela to `ContextPattern`, przechowująca wzorce kontekstu odpowiadające poszczególnym wzorcom wyrazu. Dla każdego wiersza atrybuty *l* i *r* przechowują odpowiednio treść wzorca lewego i prawego kontekstu. Atrybut *count* oznacza łączną liczbę wystąpień, a atrybut *forms* to statystyka wystąpień[14]. Atrybut *pattern* umożliwia zapis relacji wiele-do-jednego z tabelą `Pattern`.

Powyższe tabele w zupełności wystarczają do ekstrakcji wyrazów wielosegmentowych, jednak dodano jeszcze jedną tabelę, `ContextPatternBinding`. Zapisuje ona informacje o tym, które wzorce z kontekstem (atrybut *context_pattern*) powstały z danego wyrazu (atrybut *word*). Dodatkowo przechowywana jest łączna liczba i statystyka wystąpień wzorca z kontekstem dla konkretnego wyrazu. Tabela ta jest wykorzystywana m.in. w eksperymentalnym algorytmie ekstrahującym etykiety semantyczne z tekstu (podrozdział 11.2).



Rysunek 9.3: Struktura bazy danych wzorców syntaktycznych.

Przykład. Rozważmy przypadek przetwarzania czterech haseł: “Ślad macierzy”, “Cząstka elementarna”, “Łódź podwodna” i “Wojny syryjskie”. Załóżmy, że utworzono dla nich poniższe wzorce z kontekstem i ich statystyki wystąpień.

Wyraz	Wzorzec	Statystyka
ślad macierzy	`|*ac1 ad2,ad3,ad6,ad7,ad9 cc37`	$(\{1,4\},1)$
cząstka elementarna	`ac1,ac4 *ad1 *cc15 g`	$(\{2,3,6\},3),(\{9\},8)$
łódź podwodna	`ac1,ac4 *ad1 *cc15 g`	$(\{9\},1)$
łódź podwodna	`ac1,ac4,ad9 *ad1 *cc15 g`	$(\{9\},7)$
wojny syryjskie	`ac1,ac4 *ad8 *cc36 g`	$(\{9\},1)$

Zapiszmy je teraz do bazy danych. W tabeli `Pattern` pojawią się trzy wiersze:

[13] Czyli posiadającymi atrybut *is_entry* = `True`.

[14] Atrybut nazwano *forms*, ponieważ liczba wystąpień jest grupowana wg numerów form.

id	text	count
1	∗ac1 ad2,ad3,ad6,ad7,ad9	1
2	∗ad1 ∗cc15	19
3	∗ad8 ∗cc36	1

Referencje do powyższych wzorców zapisujemy w odpowiednich wierszach tabeli `Word`:

id	name	decl	tags	pattern
1	ślad macierzy	∗-	ACA-ślad:1 ADA-macierz:2,3,6,7,9	1
2	cząstka elementarna	∗∗	ADA-cząstka:1 CC-elementarny:15	2
3	łódź podwodna	∗∗	ADA-łódź:1 CC-podwodny:15	2
4	wojny syryjskie	∗∗	ADA-wojna:8 CC-syryjski:36	3

Natomiast wzorce kontekstu (tabela `ContextPattern`) wyglądają następująco:

Tabela 9.3: Wzorce kontekstu zapisane do tabeli `ContextPattern`.

id	l	r	count	forms	pattern
1	"|"	" cc37"	1	1,4:1	1
2	"ac1,ac4 "	" g"	12	2,3,6:3 9:9	2
3	"ac1,ac4,ad9"	" g"	7	9:7	2
4	"ac1,ac4 "	" g"	1	9:1	3

Wzorzec kontekstu nr 2 utworzono dla dwóch wyrazów. Obrazuje to tabela `ContextPatternBinding`:

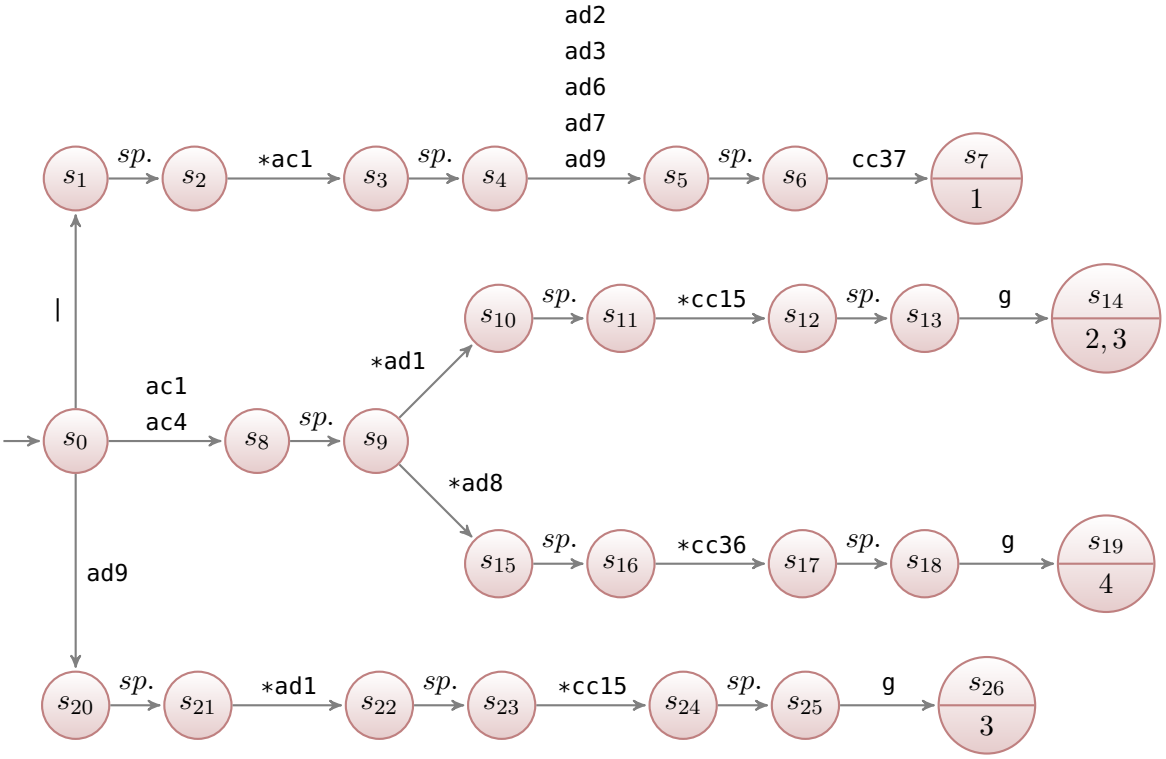
id	word	context_pattern	count	forms
1	1	1	1	1,4:1
2	2	2	11	2,3,6:3 9:8
3	3	2	1	9:1
4	3	3	7	9:7
5	4	4	1	9:1

9.4.1. Przebieg zapisu wzorców

Aby zoptymalizować proces zapisu, należy zminimalizować liczbę operacji aktualizacji bazy danych, a także grupować zapytania INSERT. Postanowiono więc podzielić proces zapisu na dwie fazy. W pierwszym kroku dla każdego wyrazu zapisywany jest wzorzec syntaktyczny wyrazu i wzorce kontekstu wraz ze statystyką wystąpień. Zapisywane są one do tymczasowych atrybutów tabeli `Word`. W kolejnym etapie dokonywana jest agregacja wzorców i tworzenie wierszy tabeli `Pattern`, `ContextPattern` i `ContextPatternResult`. Takie podejście upraszcza też zrównoleglenie algorytmu.

9.5. Rozpoznawanie wzorców syntaktycznych – metoda SM

Utworzone i zapisane do bazy danych wzorce mogą teraz posłużyć do ekstrakcji wyrazów wielosegmentowych. Wynikową metodę nazwiemy **SM** (*Syntactic pattern Matching*). W pierwszej kolejności należy utworzyć automat rozpoznający wzorce. Wykorzystujemy do tego ponownie algorytm 1 (podrozdział 7.3). Trzeba tylko zdefiniować operację podziału wzorca syntaktycznego na segmenty (linia 1 algorytmu). W tym celu każdy wzorzec segmentu musimy podzielić na składowe opcje segmentu wszędzie tam, gdzie dla danego segmentu mamy kilka różnych możliwości, które muszą być rozpatrzone indywidualnie. Okazuje się to dość proste – na rys. 9.1 i 9.2 dostrzegamy opcje oddzielone przecinkami – to właśnie na te opcje będzie dzielony wzorzec segmentu. Dla przykładu na rys. 9.4 przedstawiono automat rozpoznający wzorce z powyższego przykładu (podrozdział 9.4). Widzimy, że np. w stanie s_4 każda z opcji posiada oddzielne przejście – zostanie ona rozpoznana nawet wtedy, gdy wystąpi samodzielnie.



Rysunek 9.4: Diagram stanów automatu rozpoznającego wzorce z tab. 9.3. Przejścia wielokrotne między daną parą stanów oznaczono dla uproszczenia jedną strzałką i zapisano jedno nad drugim. Symbol *sp.* oznacza spację. Liczby pod symbolem stanu oznaczają identyfikatory rozpoznanych wzorców (wyjście automatu).

Zastanówmy się teraz, czy algorytm rozpoznawania wzorców słownikowych (DM) można wykorzystać do rozpoznawania wzorców syntaktycznych w tekście. Główny algorytm rozpoznający wzorce słownikowe (alg. 4) może posłużyć do rozpoznawania wzorców syntaktycznych po jednej drobnej modyfikacji – w linii 2 podczas generowania segmentów tekstu należy generować nowy segment "|" na początku każdego zdania, a także po całym tekście. Pozwoli to rozpoznawać poprawnie wzorce z niepełnym kontekstem. W linii 4 znaku "|" nie traktujemy jak separatora, lecz jak segment, co pozwala na rozpoznawanie

wzorców zaczynających się od tego znaku.

Przeanalizujmy teraz inne algorytmy wykorzystywane w alg. 4. Funkcje `initialize_matcher` i `match_segment` (alg. 3) pozostają aktualne, ponieważ nie ma w nich żadnych zależności od wzorców słownikowych – zostały one celowo napisane w ogólny sposób. Jedynym algorytmem, który musimy dokładniej przemyśleć, jest generowanie wariantów segmentu (alg. 2). Ponownie okazuje się, że właściwie większość algorytmu pozostaje bez zmian, a jedyne co trzeba zdefiniować ponownie, to reguły generowania opcji segmentu nieodmiennego (linia 1) i odmiennego (linia 6). Analogicznie jak dla wzorców słownikowych, tak i tutaj należy utworzyć zbiory wszystkich możliwych opcji dla danego segmentu, biorąc pod uwagę fakt, że wyraz występujący na początku zdania pisany jest wielką literą, a powinien również pasować do wzorca pisanego małą literą.

W przypadku segmentu odmiennego generujemy zbiory opcji dla trójek (s, id, f_0) (linia 6 alg. 2). Dla identyfikatora id tworzymy etykietę l wg reguły numer 1 tworzenia wzorców syntaktycznych (podrozdział 9.1). Następnie:

- Jeżeli pierwsza litera s jest mała lub (pierwsza litera s jest wielka i s jest pierwszym segmentem w zdaniu, a pierwsza litera $bform(id)$ jest mała), dodajemy opcję $*l_{lower} f_0$, gdzie l_{lower} to etykieta l pisana małymi literami.
- Jeżeli pierwsza litera s jest wielka, dodajemy opcję $*l f_0$.

Dla segmentu nieodmiennego sytuacja jest bardziej skomplikowana, ponieważ trzeba wygenerować możliwe wzorce zarówno kontekstu jak i wyrazu[15]. Najpierw rozważmy możliwość, że dla segmentu s znacznik słownikowy $tag(s)$ jest niepusty. Wówczas dla każdej z par $(id, f) \in tag(s)$ jeżeli id oznacza rzeczownik, przymiotnik bądź liczebnik, stosujemy powyższe reguły dla segmentu odmiennego, przy czym pomijamy początkową gwiazdkę, a zamiast f_0 stosujemy f. Natomiast jeżeli s jest słowem nieodmiennym (etykieta fleksyjna G), wówczas musimy wygenerować poniższe elementy:

- Dodajemy element G.
- Jeżeli s jest przyimkiem, dodajemy elementy Gn, gdzie n to oczekiwany numer przypadka rzeczownika dla tego przyimka.
- Jeżeli s nie jest przyimkiem, dodajemy element G0.
- Dodajemy segment s w cudzysłowie.

Po utworzeniu powyższych opcji musimy utworzyć wersje powyższych elementów pisane wielką lub małą literą, podobnie jak dla segmentu odmiennego – np. dla słowa “Za” występującego na początku zdania wygenerujemy zbiór {G, G4, G5, "Za", g, g4, g5, "za"}.

Dla innych wyrazów generujemy element “W”, przy czym znowu stosujemy powyższą regułę o wielkiej i małej literze. Dla liczby generujemy element “n”, a dla innego segmentu – element “o”.

[15] Podczas rozpoznawania wzorców nie wiadomo, z którym fragmentem wzorca mamy obecnie do czynienia.

9.5.1. Przebieg rozpoznawania wzorców syntaktycznych

Powyższe reguły wraz z ponownie wykorzystanymi algorytmami pozwalają na przeprowadzenie procesu rozpoznawania wzorców syntaktycznych. Przeanalizujmy więc rozpoznawanie wzorców dla przykładowego automatu przedstawionego na rys. 9.4. Dla uproszczenia, podobnie jak w podrozdziale 7.4.1, przeanalizujemy jedynie rozpoznawanie wzorców zaczynających się w jednym wybranym segmencie. Wyrażenie, w którym rozpoznajemy wzorce, to fragment zdania: “Rozwój chmur kłębiastych i lokalnych burz.”, a segment, od którego zaczynamy rozpoznawanie, to “Rozwój”. Kolejne kroki działania algorytmu przedstawiono w tab. 9.4. Zbiory V pokazują, jak dla każdego segmentu generowane są warianty segmentu – widzimy zarówno warianty odmienne ze zbiorami form, jak i nieodmienne ze zbiorem F_∞. Po przetworzeniu segmentu odmiennego “chmur” algorytm rozwidla się na dwie ścieżki, co prowadzi do rozpoznania wzorców 2 i 3 (l. poj.) oraz 4 (l. mn.). Analizę zakończono na segmencie “i”, ponieważ ze stanów s_{14} i s_{19} nie ma dalszych przejść i zbiór M_{next} jest pusty.

Tabela 9.5 pokazuje wyniki rozpoznawania wzorców dla całego zdania (a nie tylko dla opisanego przypadku z segmentem początkowym “Rozwój”). Dzięki temu, że rozważono przypadek ze specjalnym segmentem “l” oznaczającym początek zdania, pojawił się jeszcze wynik rozpoznawania wzorca nr 1 – rozpoznane wyrażenie to “Rozwój chmur”. W tym konkretnym przypadku nie jest to poprawny WW.

Tabela 9.4: Kolejne kroki działania algorytmu rozpoznającego wzorce syntaktyczne dla wyrażenia “Rozwój chmur kłębiastych i lokalnych burz.” dla segmentu początkowego “Rozwój”. Zawartość zbioru R podano w uproszczeniu – ograniczono ją do identyfikatorów wzorca.

Seg./sep.	**Zbiór wariantów** V	**Zbiór** M_{next}	R
Rozwój	(`ac1,ac4,AC1,AC4` , F_∞), (`*ac1,*AC1` , $\{1,4\}$)	(s_8, `ac1,ac4` , F_∞)	$\varnothing$
spacja	(` ` , F_∞)	(s_9, `ac1,ac4` ` ` , F_∞)	$\varnothing$
chmur	(`ad9` , F_∞), (`*ad1,*ad8` , $\{9\}$)	(s_{10}, `ac1,ac4` `*ad1` , $\{9\}$), (s_{15}, `ac1,ac4` `*ad8` , $\{9\}$)	$\varnothing$
spacja	(` ` , F_∞)	(s_{11}, `ac1,ac4` `*ad1` ` ` , $\{9\}$), (s_{16}, `ac1,ac4` `*ad8` ` ` , $\{9\}$)	$\varnothing$
kłębiastych	(`cc30,ad32,cc34,cc37,cc41` , F_∞), (`*cc1,*cc29` , $\{9,11,13\}$), (`*cc8,*cc15,*cc22,*cc36` ,$\{9,13\}$)	(s_{12}, `ac1,ac4` `*ad1` `*cc15` , $\{9\}$), (s_{17}, `ac1,ac4` `*ad8` `*cc36` , $\{9\}$)	$\varnothing$
spacja	(` ` , F_∞)	(s_{13}, `ac1,ac4` `*ad1` `*cc15` ` ` , $\{9\}$), (s_{18}, `ac1,ac4` `*ad8` `*cc36` ` ` , $\{9\}$)	$\varnothing$
i	(`"i",g,g0` , F_∞)	$\varnothing$	2, 3, 4

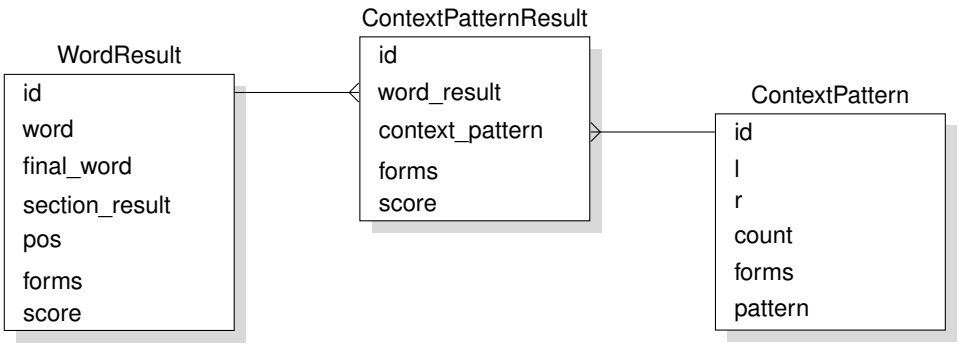
Tabela 9.5: Wyniki (zawartość zbioru R) rozpoznawania wzorców syntaktycznych dla wyrażenia "Rozwój chmur kłębiastych i lokalnych burz.". Dla każdego wyniku podano również rozpoznane wyrażenie.

Id. wzorca	Ścieżka ($Path$)	Formy (F)	Wyrażenie
1	| *ac1 ad9 cc37	1, 4	Rozwój chmur
2, 3	ac1,ac4 *ad1 *cc15 g	9	chmur kłębiastych
4	ac1,ac4 *ad8 *cc36 g	9	chmur kłębiastych

9.6. Przetwarzanie i zapis wyników

Zapis wyników ekstrakcji wygląda podobnie, jak dla algorytmu DM. Wykorzystujemy ponownie strukturę bazy danych opisaną w podrozdziale 7.5.1 i zilustrowaną na rys. 7.5. Zapisywanie sekcji w tabeli `Section` i wyrazów wielosegmentowych w tabeli `Word` przebiega bez zmian. W przypadku wyniku sekcji (tabela `SectionResult`) pojawiają się już pewne różnice. Dotyczą one wyboru optymalnego podzbioru wyników sekcji – tym razem okazało się, że reguła odrzucająca wyniki zawierające się w innych pogarsza uzyskane rezultaty – przykładowo, jeżeli otrzymamy wyniki "rekordowy skok w dal" i "skok w dal", odrzucenie tego drugiego byłoby błędem. Przyczyną jest fakt, że dla SM wykrywamy wyrazy pasujące do wzorca, a nie wyrazy ze słownika. Wybór powinien więc w całości opierać się na maksymalnej sumie **ocen** (atrybut *score*) poszczególnych wyników. Kolejną ważną tabelą zawierającą wyniki metody SM jest `WordResult`. Podobnie jak dla DM, łączy ona relacją wiele-do-wielu wyrazy wielosegmentowe i wyniki sekcji. Tutaj również występuje atrybut *pos*, który otrzymuje wartość 0 dla wyniku wybranego i 1 dla pozostałych. Wybór następuje w oparciu o ocenę (atrybut *score*).

Istotną nowością w stosunku do metody DM jest wprowadzenie nowej tabeli **wyników wzorca** o nazwie `ContextPatternResult` (rys. 9.5), przechowującej informacje o rozpoznanych wzorcach z kontekstem dla każdego wyniku wyrazu. Tabelę `ContextPatternResult` z `WordResult` łączy więc relacja wiele-do-jednego. Dla każdego wyniku wzorca przechowujemy referencję do rozpoznanego wzorca z kontekstem (atrybut *context_pattern*), ref. do wyniku wyrazu (*word_result*), listę rozpoznanych numerów form wyrazu (*forms*) oraz ocenę (*score*). Przeanalizujmy teraz operację wyznaczania oceny dla każdego wyniku wzorca. Jest to wykonywane w oparciu o wartości atrybutów *forms* tego wyniku oraz wzorca



Rysunek 9.5: Nowa tabela `ContextPatternResult`, łącząca wyniki wyrazu z wzorcami syntaktycznymi.

z kontekstem. Przypominamy, że dla wzorca atrybut ten przechowuje statystykę wystąpień wzorca w poszczególnych formach w postaci listy par (G, N_G). Ocena powinna uwzględniać te wystąpienia, które odpowiadają rozpoznanym formom, będzie to więc suma tych wartości N_G, dla których przecięcie G i atrybutu *forms* wyniku jest niepuste. Przykładowo, dla statystyki przedstawionej w tab. 9.1 i wyniku, którego atrybut *forms* zawiera formy 5 i 6, ocena będzie miała wartość 4[16].

Możemy teraz ostatecznie zdefiniować ocenę **wyniku wyrazu** jako sumę ocen odpowiadających mu wyników wzorca oraz ocenę **wyniku sekcji** jako ocenę wybranego wyniku wyrazu. Podobnie jak w przypadku metody DM, wyniki, które nie zostały wybrane, mogą zostać pominięte przy zapisie – są one domyślnie zapisywane, aby umożliwić późniejszą analizę dokonanych wyborów.

9.6.1. Proces zapisu wyników

Podobnie jak dla metody DM, zapis wyników jest dość skomplikowany ze względu na złożoność relacji między tabelami – w tym przypadku przybyła jeszcze jedna tabela. Przeanalizujemy ten proces na przykładzie wyników z tabeli 9.5. W pierwszej kolejności zapiszmy wyekstrahowane wyrazy w tabeli `Word`[17] – w tym przypadku rozpoznano trzy różne wyrazy wielosegmentowe, odpowiadające kolejnym wierszom tab. 9.5[18]:

id	name	decl	tags
1	rozwój chmur	`*-`	`ACA-rozwój:1 ADA-chmura:9`
2	chmura kłębiasta	`**`	`ADA-chmura:1 CC-kłębiasty:15`
3	chmury kłębiaste	`**`	`ADA-chmura:8 CC-kłębiasty:36`

W tabeli `Section` znajdzie się jeden wiersz, ponieważ wszystkie rozpoznane wyrażenia mają część wspólną:

id	start	end	text
1	0	3	Rozwój chmur kłębiastych

W sekcji tej znajdziemy dwa możliwe rozpoznane wyrażenia – w tabeli `SectionResult` zapiszemy dwa wiersze. Drugi wynik ma wyższą ocenę, zostaje więc wybrany (*pos* = 0).

id	section	pos	start	end	text	score
1	1	1	0	2	Rozwój chmur	1
2	1	0	1	3	chmur kłębiastych	16

Tabela `WordResult` zawiera 3 wiersze, ponieważ drugiemu wynikowi sekcji odpowiadają dwa możliwe wyrazy. Spośród opcji "chmura kłębiasta" i "chmury kłębiaste" wybierzemy tę pierwszą ze względu na wyższą ocenę.

[16] Zsumowano wartości N_G dla par $(\{5, 6\}, 3)$ i $(\{6\}, 1)$.

[17] Pomijamy szczegóły algorytmu wyznaczającego wartości poszczególnych atrybutów.

[18] Rozpatrywany przykład jest prosty – wyniki ze zbioru R są w relacji jeden-do-jednego z rozpoznanymi wyrazami. W rzeczywistości nie zawsze tak będzie – w szczególności nawet pojedyncze wystąpienie wyrazu może zostać rozpoznane wielokrotnie.

id	word	section_result	pos	forms	score
1	1	1	0	1, 4	1
2	2	2	0	9	16
3	3	2	1	9	1

W tabeli `ContextPatternResult` są cztery wiersze, po jednym dla każdego rozpoznanego wzorca (każdy z nich rozpoznano 1 raz). Widzimy też, że ocena 16 jest tak naprawdę sumą ocen dwóch wzorców. Tabelę `ContextPattern` z wzorcami przedstawiono w tab. 9.3.

id	word_result	context_pattern	forms	score
1	1	1	1, 4	1
2	2	2	9	9
3	2	3	9	7
4	3	4	9	1

Podsumowując powyższy przykład możemy stwierdzić, że na koniec dzięki najlepszej ocenie wybrany został wynik "chmura kłębiasta". W opisywanym przypadku jest to rzeczywiście poprawny wybór, jednak ciężko oczekiwać, by tak było zawsze – metody statystyczne zawsze obarczone są pewnym błędem, a wybór nie zawsze jest oczywisty, nawet dla człowieka.

9.7. Testy metody SM

Opisana w niniejszym podrozdziale metoda SM została przetestowana na próbce testowej PAP-WW. Metoda ta w przeciwieństwie do DM posiada kilka parametrów liczbowych, które wpływają na uzyskany wynik.

– W podrozdziale 8.1.5 opisano wyznaczane dla każdego wyrazu (wiersza tabeli `Word`) współczynniki jakości: atrybuty *score* i *int_score*. Po wypełnieniu tabeli `Word` wyrazami można więc w oparciu o wartości tych współczynników wybrać do tworzenia wzorców tylko słowa o jakości powyżej pewnego progu. Okazało się, że najbardziej użyteczny jest parametr będący iloczynem *score* i *int_score*. Oznaczmy jego wartość progową przez ws_{min}. Tylko wyniki o iloczynie *score* i *int_score* większym lub równym tej wartości zostaną wzięte pod uwagę przy tworzeniu wzorców. Domyślnie $ws_{min} = 0$.

– Po zapisaniu wzorców w tabeli `Pattern` można posłużyć się atrybutem *count* tych wzorców, biorąc przy tworzeniu automatu pod uwagę tylko wzorce mające liczbę wystąpień większą lub równą pewnemu progowi pc_{min}. Domyślna wartość progu to 1.

– Już po wykonaniu testu można z kolei posłużyć się ocenami poszczególnych wyników (atrybut *score* w tabeli `WordResult`), odrzucając wyniki z oceną niższą od ustalonego progu rs_{min}. Domyślna wartość tego progu to 1.

– Opisana już wcześniej długość lewego i prawego kontekstu ma bardzo duży wpływ na uzyskane wyniki. Domyślną metodę z kontekstami o długości 1 oznaczamy przez $SM_{1,1}$.

Dobór parametrów liczbowych jest pewną formą optymalizacji metody, więc gdyby optymalizacja ta była przeprowadzana dla jednej próbki testowej, istniałoby niebezpieczeństwo przekłamania wyników przez nadmierne dopasowanie do próbki testowej (*overfitting*). Dlatego też zastosowano następującą metodę:

1. Korpus testowy PAP-WW był dzielony na dwie połowy po 50 notatek prasowych.
2. Parametry optymalizowane były osobno dla każdej z połówek.
3. W przypadku braku różnicy w wartościach optymalnych przyjmowano je za zwalidowane.
4. W przypadku różnic w wartościach optymalnych test był wykonywany osobno dla każdej połowy PAP-WW, przy czym parametry optymalizowane dla pierwszej części testowane były na drugiej i odwrotnie (walidacja krzyżowa).

Ta procedura używana była również dla metod opisywanych w kolejnym rozdziale. Co ważne, potrzeba wykonania kroku 4 zachodziła niezmiernie rzadko, a ewentualne różnice były mniejsze niż 0.1%. Świadczy to o dużej wiarygodności testu.

Wyrazy z wzorcami odmiany wyznaczone zostały wyznaczone analogicznie, jak dla testów pDM (podrozdział 8.3), czyli do wyznaczania wzorca odmiany użyto linków i wystąpień hasła w treści artykułu – również dla SM dodatkowe przetwarzanie treści artykułu dało pozytywny rezultat. Wyrazów z wyznaczonymi wzorcami odmiany jest 176506 (tyle samo, co dla testu pDM), a 173763 z nich zawiera jednoznaczne wzorce odmiany[19].

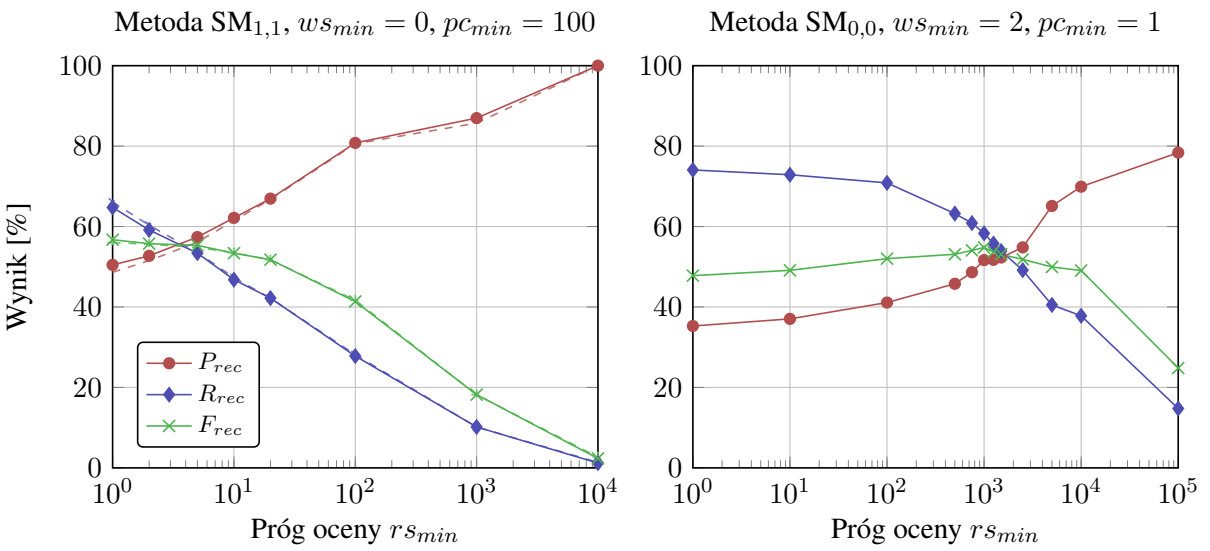
Wyniki testów rozpoznawania wzorców metodą SM przedstawiono w tab. 9.6 – dla każdej metody podano optymalne wartości parametrów ze względu na F_{rec}. Najlepsze wyniki dała przedstawiona w pierwszym wierszu metoda $SM_{1,1}$, czyli SM z długością lewego i prawego kontekstu równą 1[20]. W przypadku tego wariantu nie tylko wartość F_{rec} była najwyższa, ale również dobór wartości parametrów

[19] Atrybut *decl* nie zawiera dla nich znaku "?".

[20] Przetestowano długości lewego i prawego kontekstu równe odpowiednio $(0,0)$, $(0,1)$, $(1,0)$, $(1,1)$, $(1,2)$ i $(2,1)$.

Tabela 9.6: Wyniki testów rozpoznawania i ekstrakcji wyrazów wielosegmentowych różnymi wariantami metody SM. Dla każdego wariantu dobrano wartości parametrów dające największą wartość F_{rec}.

	Opt. wartości parametrów			**Test rozpoznawania**			**Test ekstrakcji**		
Metoda	ws_{min}	pc_{min}	rs_{min}	P_{rec}	R_{rec}	F_{rec}	P_{ext}	R_{ext}	F_{ext}
$SM_{1,1}$	0	100	1	50.46	64.75	**56.72**	47.82	61.36	53.75
$SM_{0,0}$	2	1	1000	51.65	58.31	54.78	48.95	55.25	51.91
$SM_{W1,1}$	2	3000	1	48.91	64.92	55.79	47.25	62.71	**53.90**
$SM_{W0,0}$	2	7000	2000	51.84	59.83	55.55	49.93	57.63	53.50



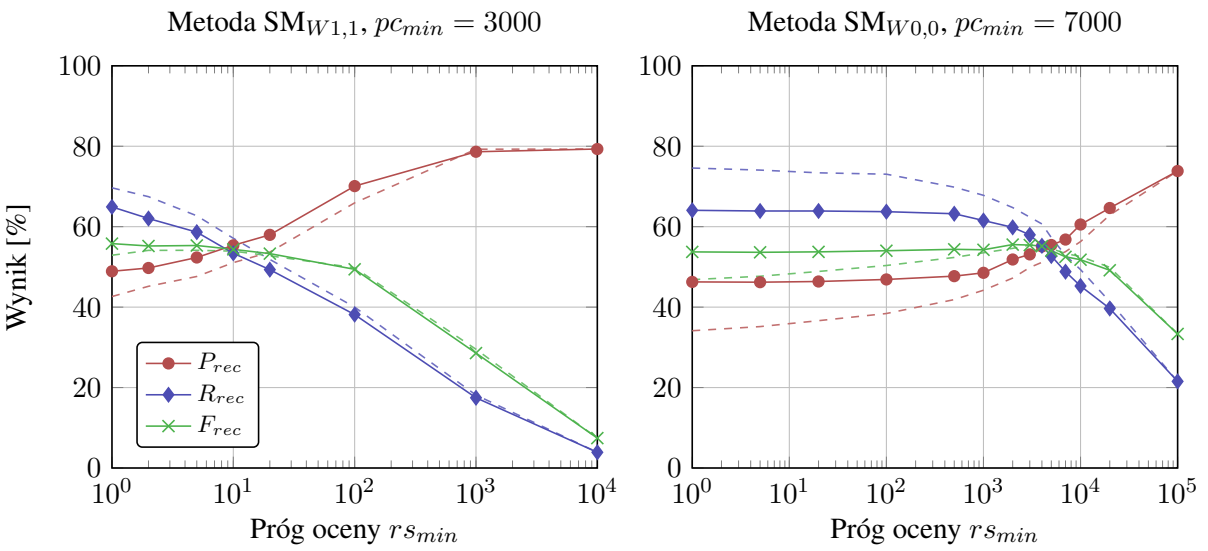
Rysunek 9.6: Wyniki testu rozpoznawania WW metodą SM$_{1,1}$ i SM$_{0,0}$ w zależności od progu rs_{min}. Dodatkowo dla wariantu SM$_{1,1}$ przerywaną linią oznaczono wyniki dla domyślnej wartości $pc_{min} = 1$.

był najłatwiejszy. Dla domyślnych wartości $ws_{min} = 0$, $pc_{min} = 1$ i $rs_{min} = 1$ uzyskano już wartość F_{rec} powyżej 56%. Wzorców wyrazu (wierszy tabeli `Pattern`) było 30764, natomiast wzorców kontekstu (wierszy tabeli `ContextPattern`) było 560994. Na rys. 9.6 zaprezentowano wyniki testu rozpoznawania WW w zależności od rs_{min}. Zwiększanie progu prowadzi do poprawy precyzji, lecz jednocześnie pomijanych jest coraz więcej poprawnych wyników – spada więc pełność. Metoda SM$_{1,1}$ osiąga najwyższą wartość F_{rec} dla $rs_{min} = 1$ (funkcja jest malejąca), co sugeruje, że być może ograniczeniem metody jest niska gęstość wzorców – sieć linków Wikipedii jest jednak dość rzadka. Niewielką dodatkową poprawę udało się osiągnąć zwiększając próg pc_{min} do 100, czyli odrzucając rzadko występujące wzorce. Dodatkowo operacja ta pozwala zmniejszyć znacznie liczbę wzorców, minimalizując czas działania i zużycie pamięci.

Drugim obok SM$_{1,1}$ wariantem, który dał wartościowe wyniki, jest wariant bez kontekstu, czyli SM$_{0,0}$. Na rys. 9.6 widzimy, że dla $rs_{min} = 1$ wynik jest tutaj zupełnie inny – wysoka pełność okupiona jest obniżoną precyzją, a także niższą wartością F_{rec}. Trzeba zaznaczyć, że zastosowano tutaj wartość $ws_{min} = 2$, czyli odfiltrowano wyrazy o niskiej jakości wzorca odmiany. Bez tego zabiegu wyniki były dużo gorsze. Zwiększając rs_{min} możemy dojść do wartości F_{rec} tylko o ok. 2% gorszej od SM$_{1,1}$, jednak wyraźnie widać, że maksimum występuje dla wąskiego przedziału rs_{min}, więc być może dla innego korpusu testowego będzie ono położone gdzie indziej. Ponadto SM$_{1,1}$ uzyskuje dużo lepszy wynik dla P_{rec} = 80% – wartość F_{rec} przekracza 40%, podczas gdy dla SM$_{0,0}$ wynosi niewiele ponad 20%.

Podobnie jak dla pDM, tak i tutaj wykonano również eksperyment polegający na tym, żeby podczas tworzenia wzorców odmiany wziąć pod uwagę wystąpienia hasła w korpusie WIKI (por. podrozdział 8.3). Zmodyfikowanym metodom dodano oznaczenie W, a ich wyniki przedstawiają dwa dolne wiersze tab. 9.6. Na rys. 9.7 widzimy z kolei zależność wyników od progu oceny. W przypadku obu metod obserwujemy zwiększenie pełności[21], jednak jednocześnie precyzja dość znacznie spada, mimo że dla obu metod

[21] Zwłaszcza dla oznaczonych liniami przerywanymi wykresów dla $pc_{min} = 1$.



Rysunek 9.7: Wyniki testu rozpoznawania WW metodą SM$_{W1,1}$ i SM$_{W0,0}$ dla $ws_{min} = 2$ w zależności od progu rs_{min}. Dodatkowo linią przerywaną oznaczono wyniki dla domyślnej wartości $pc_{min} = 1$.

stosujemy tutaj próg $ws_{min} = 2$ dla dodatkowego podniesienia precyzji[22]. Optymalne wartości F_{rec} zostały uzyskane dla mocno zwiększonej wartości pc_{min} (patrz tab. 9.6). Oznacza to, że pozostawiamy jedynie najczęściej występujące wzorce – liczba branych pod uwagę wierszy tabeli `Pattern` to 102 dla $pc_{min} = 3000$[23] oraz zaledwie 50 dla $pc_{min} = 7000$. Najbardziej interesujący jest wynik SM$_{W0,0}$, który jest dość wysoki mimo bardzo małej liczby wzorców. Niestety uzyskanie maksimum wymaga optymalizacji wartości wielu parametrów, a rys. 9.7 pokazuje, że występuje ono dla bardzo wąskiego przedziału rs_{min}.

Podsumowując uzyskane wyniki, metoda SM opierająca się na rozpoznawaniu wzorców syntaktycznych pozwala na podniesienie wartości R_{rec} nawet do ponad 70%, dzieje się to jednak kosztem znaczącego spadku precyzji. Jest tak dlatego, ponieważ o tym, czy dane słowo jest wyrazem wielosegmentowym, decyduje nie tylko jego struktura syntaktyczna, ale także informacje semantyczne, które w tej metodzie nie są uwzględniane[24]. Można jednak posłużyć się metodą SM w celu zbudowania **słownika wyrazów wielosegmentowych** – koncepcja ta zostanie bliżej przedstawiona w kolejnym rozdziale.

[22] Bez tego zabiegu dopuszczamy wyrazy nie posiadające w ogóle linków przychodzących – prowadzi to do dużego spadku precyzji.

[23] Wzorców kontekstu dla SM$_{W1,1}$ jest jednak w dalszym ciągu 134286.

[24] Można by też rozbudować strukturę wzorców lub użyć metod uczenia z nadzorem do poprawy skuteczności – rozwiązania te przedyskutowano w rozdziale 11.

10. Metoda SDM

Ekstrakcja WW w oparciu o rozszerzony słownik

Rozpoznawanie wyrazów wielosegmentowych przeprowadzane z użyciem słownika (metody DM i pDM) uzyskuje niską wartość wskaźnika *recall* ze względu na niekompletność tego słownika. Pojawia się więc pomysł, by rozszerzyć słownik o wyniki uzyskane przy pomocy metody SM zastosowanej do dużego korpusu tekstu. W ten sposób wynikowa tabela `Word` zawierałaby słownik, który można by następnie użyć w metodzie pDM. Pozostaje do rozwiązania problem wyboru korpusu źródłowego i selekcji wyrazów o wysokiej jakości. Ponadto okazuje się, że wyniki metody SM są czasem niejednoznaczne i mogą wymagać wstępnego przetwarzania przed utworzeniem słownika.

10.1. Tworzenie słownika WW z wyników SM

Jako korpus tekstu do tworzenia słownika wyrazów wielosegmentowych zdecydowano się zastosować utworzony już wcześniej korpus WIKI zawierający treść całej Wikipedii. Ponadto przeprowadzono też eksperyment polegający na wykorzystaniu korpusu PAP z pominięciem próbki testowej PAP-WW – oznaczmy go przez PAP-TRAIN. Korpus ten mimo swojej niewielkiej objętości zawiera teksty w stylu zbliżonym do próbki testowej.

Mając już dany korpus tekstów, ekstrahujemy zawarte w nim WW przy pomocy metody SM. Warto zauważyć, że jedyne tabele, których potrzebujemy, to `Word` (z wyekstrahowanymi wyrazami) i `WordResult` (z ocenami poszczególnych wyników). Pozostałych tabel można więc w ogóle nie wypełniać danymi – przyspieszy to znacznie proces przetwarzania danych dla dużego korpusu oraz pozwoli oszczędzić pamięć. Dodatkowo cały proces jest zrównoleglony, aby przebiegał szybciej. Należy też dokonać wyboru parametrów metody SM – zdecydowano się na przetestowanie kilku możliwości, które najlepiej spisały się w teście na korpusie testowym (tab. 9.6).

Kolejnym krokiem jest ujednoznacznienie wynikowych wyrazów, aby udało się z nich utworzyć słownik. W tym celu wynikowe wyrazy (wiersze tabeli `Word`) są grupowane według ich formy podstawowej (atrybut *name*). Następnie dla każdej z takich grup należy utworzyć jedno hasło wynikowego słownika. Poniżej przedstawiono przykładową listę wyrazów dla formy podstawowej “prymas Polski” wraz z odpowiadającymi im wynikami wyrazu.

nr	Atrybut *decl*	Atrybut *tags*	Liczba wyników	Σ ocen wyników
1	`*-`	`AAA-prymas:1 AAA-Polski:1,7|ADA-Polska :2,8,11,14|CC-polski:1,7,8,11,14`	415	9856
2	`**`	`AAA-prymas:1 AAA-Polski:1|CC-polski:8`	18	166
3	`**`	`AAA-prymas:1 CC-polski:8`	42	144
4	`**`	`AAA-prymas:1 AAA-Polski:1`	27	120
5	`*-`	`AAA-prymas:1 ADA-Polska:2`	25	46
6	`*-`	`AAA-prymas:1 CC-polski:1,7,8,11,14`	2	5

Wyrazy te są niezgodne ze sobą – w przypadku wyrazów 1, 5 i 6 segment “Polski” jest nieodmienny, a dla wyrazów 2, 3 i 4 – odmienny. Ponadto niektóre znaczniki słownikowe również się wykluczają, np. znaczniki wyrazów 1 i 5 zawierają rzeczownik “Polska”, a wyrazu 6 już nie. Testy wykazały, że obecność takich sprzecznych wyrazów w słowniku obniża jakość ekstrakcji WW przy jego pomocy, więc opracowano metodę pozwalającą na scalenie takich wyrazów.

Zauważmy, że tak naprawdę problem sprowadza się do scalenia pewnego zbioru par $(decl, tags)$ w jedną. Sytuacja okazuje się podobna do problemu znalezienia wzorca odmiany WW – w podrozdziale 8.1 wprowadzono pojęcie wzorca odmiany jako pary $\omega = (tags, decl)$ i przedstawiono algorytm umożliwiający wyznaczenie optymalnego przecięcia wzorców odmiany. W obecnym przypadku należy wyznaczyć takie przecięcie wzorców odmiany wyrazów, dla których łączna suma ocen będzie największa[1]. Dlatego też można ponownie wykorzystać algorytm przeglądający drzewo podzbiorów zbioru wszystkich rozpatrywanych wzorców odmiany (podrozdział 8.1.4)[2].

Po wyznaczeniu wartości atrybutów *tags* i *decl* wynikowego słowa można zapisać je do bazy danych. W tym celu tworzymy nowy wiersz w tabeli `Word` z atrybutem *is_final* = `true`. Sumaryczną ocenę zapisujemy w atrybucie *int_score*[3]. Dodatkowo wynikom wyrazów (tabela `WordResult`) należących do wybranego podzbioru dodajemy atrybut *final_word* będący referencją do nowo utworzonego wyrazu (patrz podrozdział 7.5.1 i diagram na rys. 7.5). Rozszerzy to możliwości późniejszej analizy zawartości słownika.

Przykład. Wyznaczmy wynikowy wyraz na podstawie powyższej tabeli dla formy tekstowej “prymas Polski”. Ponieważ algorytm wyznaczania optymalnego przecięcia wyrazów przebiega analogicznie do algorytmu przedstawionego w podrozdziale 8.1.4, przedstawimy jedynie wynik jego działania. Algorytm wybierze wyrazy 1 i 5 o łącznej wartości sumy ocen 9902. W zbiorze tym nie mogą znaleźć się wyrazy 2, 3 i 4, ponieważ mają drugi segment oznaczony jako odmienny. Z kolei wyraz 6 nie posiada identyfikatora `ADA-Polska`. Wszystkie pozostałe podzbiory mają niższe sumy ocen – najbliżej optymalnego wyniku jest podzbiór $\{1, 6\}$ z łączną sumą ocen 9861. Wyznaczmy teraz atrybuty wyrazu docelowego jako przecięcie par $(decl, tags)$ wyrazów 1 i 5. Atrybut *decl* ma oczywiście wartość "`*-`",

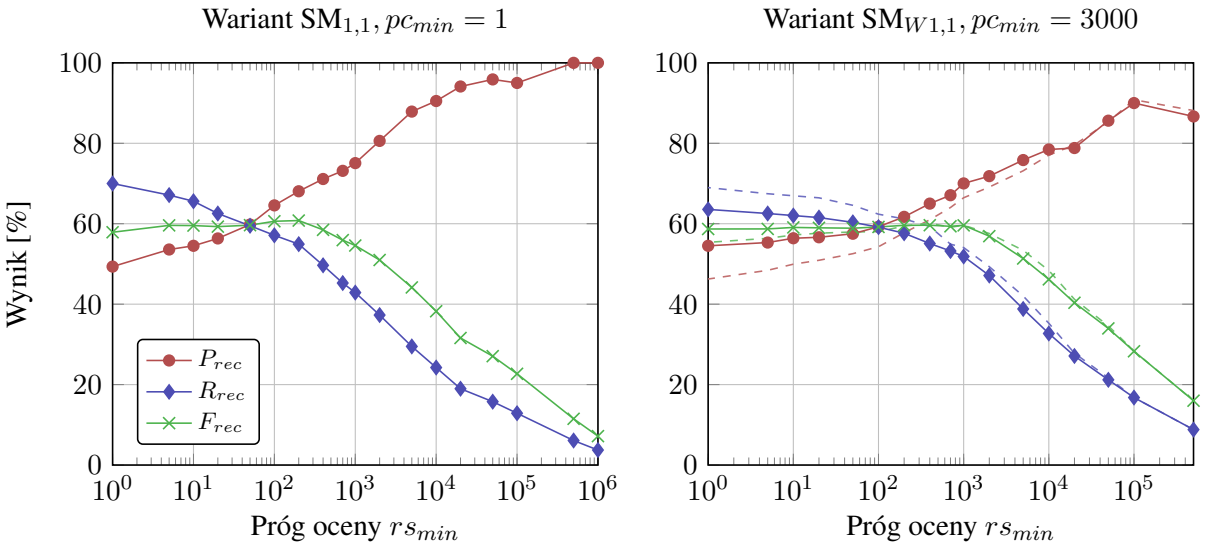
[1] Przetestowano też wersję optymalizującą łączną liczbę wyników, lecz dawała ona gorszy rezultat.
[2] Algorytm ten został dodatkowo zoptymalizowany pod kątem tego, że atrybuty *decl* są jednoznaczne.
[3] Zdecydowano się na ponowne użycie tego atrybutu, ponieważ w tym przypadku również opisuje on jakość wyrazu.

czyli segment "prymas" jest odmienny, a "Polski" – nieodmienny. Z kolei atrybut *tags* ma wartość $[\{(\texttt{AAA-prymas}, 1)\}, \{(\texttt{AAA-Polska}, 2)\}]$, czyli w postaci tekstowej `AAA-prymas:1 ADA-Polska:2`. Ujednoznacznienie identyfikatora słowa "Polska" umożliwił wyraz nr 5, którego identyfikator od początku był jednoznaczny. Czasami jednak zdarza się tak, że wszystkie wyrazy mają identyfikatory niejednoznaczne, natomiast wyznaczenie ich przecięcia pozwala na ich ujednoznacznienie – jest to kolejny argument za scalaniem wyrazów w słowniku.

10.2. Testy metody SDM

Po utworzeniu słownika wyrazów wielosegmentowych można traktować go identycznie, jak w przypadku metody pDM – z odpowiednich wierszy tabeli `Word` tworzone są wzorce słownikowe, a następnie konstruowany jest automat rozpoznający te wzorce. Cały proces staje się więc dwufazowy – faza SM generuje słownik, a faza DM używa go do ekstrakcji wyrazów. Powstałą metodę nazwano **SDM**. Ważną jej cechą jest fakt, że wyrazy słownika utworzonego przy pomocy fazy SM mają przypisaną wartość liczbową oceny (atrybut *int_score*). Pozwala to na użycie tej wartości w fazie DM do oceny poszczególnych wyników wyrazu (tabela `WordResult`), czyli rozpoznanie wyrazu o *int_score* równym s spowoduje przypisanie wynikowi również oceny s. Dzięki temu wzrasta dokładność wyboru wyniku w przypadku pokrywających się wyników sekcji (por. podrozdział 7.5.1). Podczas testów okazało się, że gdyby pominąć oceny, wyniki byłyby gorsze o około 2-3%, więc jest to istotny element metody.

Wartości ocen można też wykorzystać podobnie jak w metodzie SM, wprowadzając wartość progową oceny rs_{min}, poniżej której wyniki będą odrzucane. Alternatywnie próg rs_{min} można zastosować już na etapie konstrukcji automatu, biorąc pod uwagę jedynie wyrazy o wartości atrybutu *int_score* wyższej od



Rysunek 10.1: Wyniki testu rozpoznawania WW metodą SDM$_P$. Do budowania słownika (faza SM) użyto dwóch wariantów metody SM podpisanych powyżej wykresów. Dodatkowo dla wariantu SM$_{W1,1}$ przerywaną linią oznaczono wyniki dla $pc_{min} = 1$.

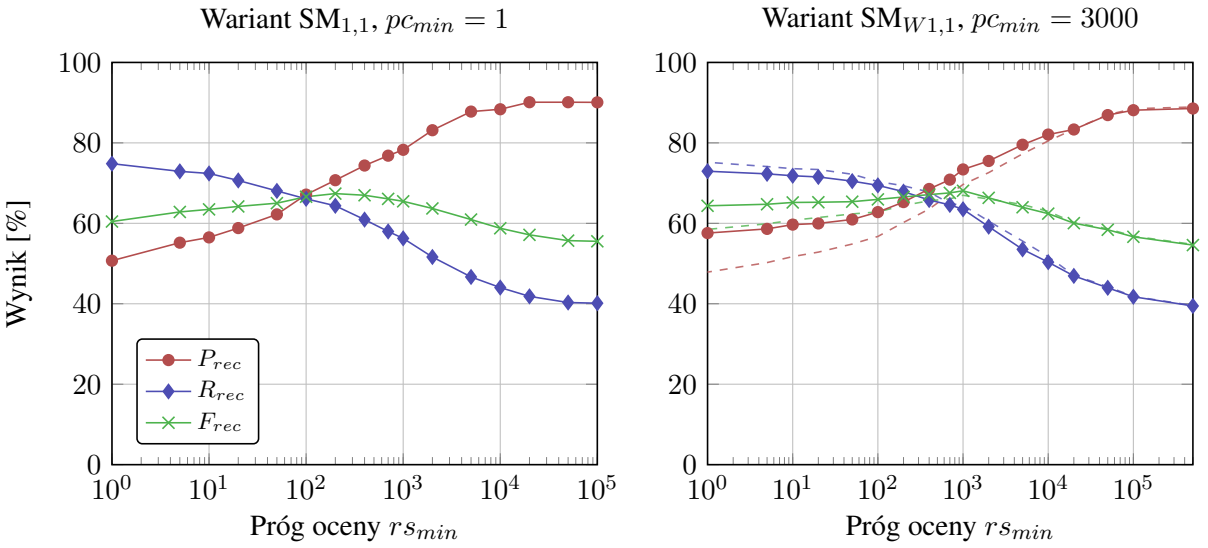
tego progu – ta opcja pozwala na znaczne zmniejszenie rozmiaru generowanego automatu[4].

W pierwszej kolejności przeprowadzono testy, wykonując fazę SM na korpusie PAP-TRAIN. Wariant ten oznaczmy przez SDM$_P$. Korpus PAP-TRAIN ma nieduży rozmiar (3.6M segmentów), przez co faza SM wykonuje się na nim dużo krócej niż dla korpusu WIKI (202.7M segmentów), więc postanowiono dokonać na nim wstępnej selekcji wariantów metody SM. Przy okazji umożliwiło to sprawdzenie, czy parametry dobrane dla specyficznego i małego korpusu okażą się poprawne dla dużo większego zasobu. Do budowania słownika posłużyły kolejno wszystkie cztery warianty metody SM z tab. 9.6 z różnymi wartościami parametru pc_{min}. Spośród czterech wariantów SM najlepsze wyniki dały warianty SM$_{1,1}$ i SM$_{W1,1}$. Na rys. 10.1 przedstawiono wyniki uzyskane dla tych metod. Sprawdzono kilka możliwych wartości parametru pc_{min} – nie była wykonywana dokładniejsza optymalizacja. Dla wariantu SM$_{1,1}$ najlepszy wynik uzyskano dla domyślnej wartości $pc_{min} = 1$, natomiast w przypadku SM$_{W1,1}$ optymalna okazała się ponownie wartość progowa równa 3000. Świadczy to o dobrej stabilności i powtarzalności wyników. Z wykresów na rys. 10.1 wynika, że nieco lepsze rezultaty dał wariant SM$_{1,1}$ – potwierdza to poprzednie testy SM. Wariant SM$_{W1,1}$ ma nieco niższą wartość F_{rec} i nie uzyskuje tak wysokiej precyzji.

Przedstawione wyniki nie są zbyt wysokie – osiągnięto niewielki wzrost F_{rec} w stosunku do wartości dla testu SM. Okazuje się jednak, że można je poprawić, łącząc metodę pDM z SDM – można to zrobić scalając słowniki, automaty lub wyniki. Zastosowano ostatni wariant, ponieważ umożliwia on zastosowanie priorytetów – w pierwszej kolejności liczyć się powinny wyniki pDM, ponieważ metoda ta osiąga wyższą precyzję. Rezultat połączonej metody pDM + SDM$_P$ zaprezentowano na rys. 10.2.

Wynik jest wyraźnie lepszy, a co ciekawe wariant SM$_{W1,1}$ okazuje się teraz równie dobry jak SM$_{1,1}$. W pierwszej części tabeli 10.1 przedstawiono cztery opisane warianty. Wartość rs_{min} dobrano tak, by

[4]Wyniki obu wariantów mogą się nieznacznie różnić, ponieważ odcięcie wyników następuje odpowiednio przed i po przeprowadzeniu wyboru optymalnego zbioru wyników w obrębie danej sekcji. Różnice są jednak niewielkie (poniżej 0.5%).



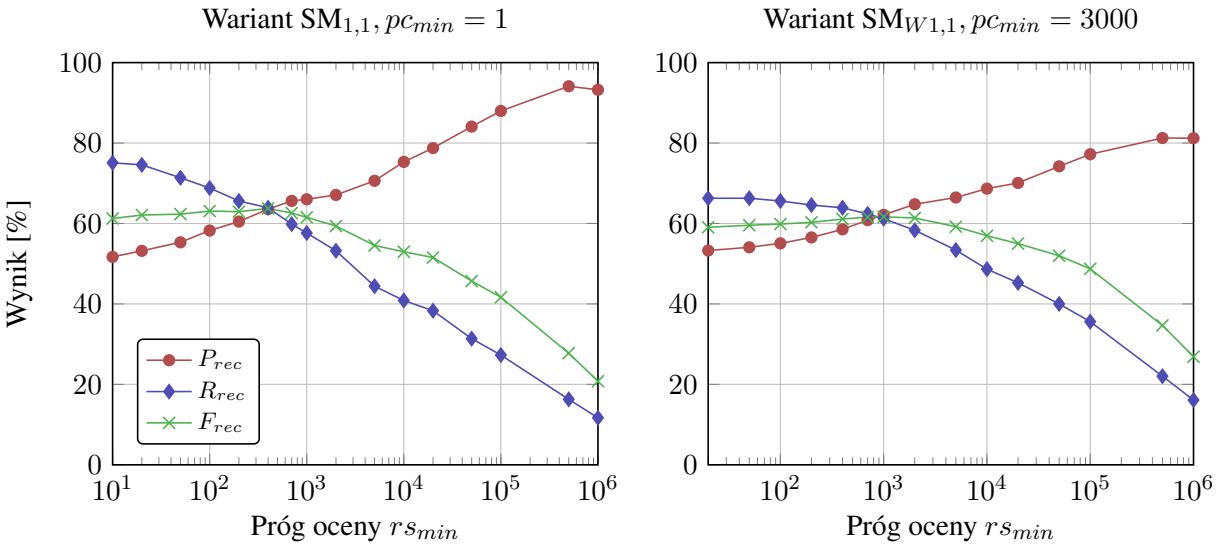
Rysunek 10.2: Wyniki testu rozpoznawania WW metodą pDM + SDM$_P$. Do budowania słownika (faza SM) użyto dwóch wariantów metody SM podpisanych powyżej wykresów. Dodatkowo dla wariantu SM$_{W1,1}$ przerywaną linią oznaczono wyniki dla $pc_{min} = 1$.

Tabela 10.1: Wyniki testów rozpoznawania i ekstrakcji wyrazów wielosegmentowych różnymi wariantami metody SDM. Dla każdego wariantu dobrano wartości parametrów dające najwyższą wartość F_{rec}.

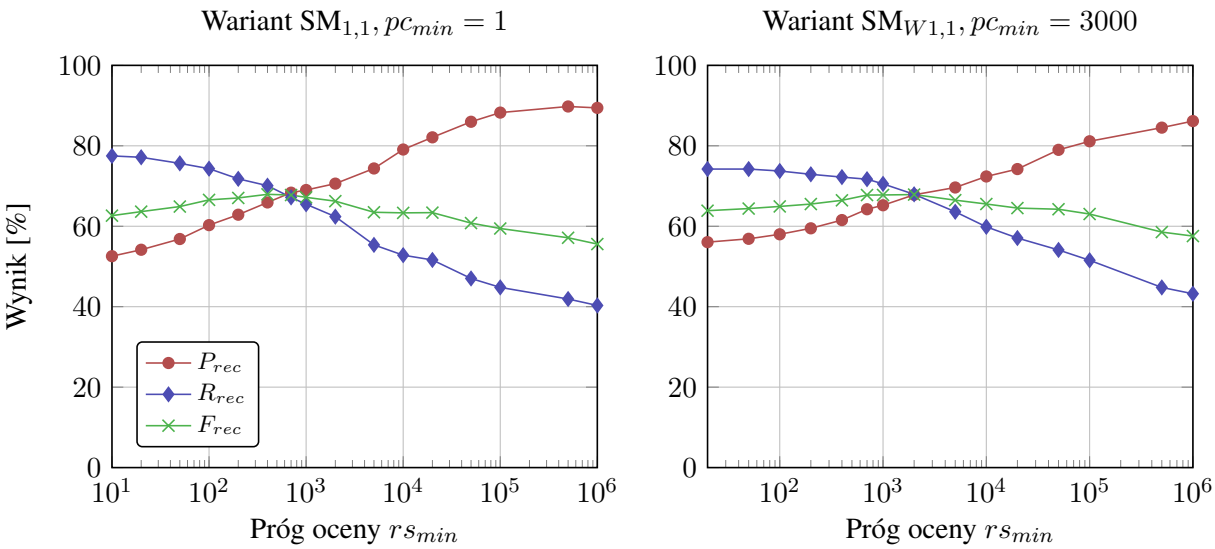
Użyta metoda			**Test rozpoznawania**			**Test ekstrakcji**		
Metoda	**Wariant SM**	rs_{min}	P_{rec}	R_{rec}	F_{rec}	P_{ext}	R_{ext}	F_{ext}
SDM_P	$\text{SM}_{1,1}$	200	68.07	54.92	60.79	65.55	52.88	58.54
SDM_P	$\text{SM}_{W1,1}$	300	63.64	56.95	60.11	61.74	55.25	58.32
$\text{pDM} + \text{SDM}_P$	$\text{SM}_{1,1}$	350	74.49	62.82	68.16	71.43	60.24	65.36
$\text{pDM} + \text{SDM}_P$	$\text{SM}_{W1,1}$	1000	73.40	63.49	68.09	70.80	61.25	65.68
SDM_W	$\text{SM}_{1,1}$	300	62.83	64.75	63.77	60.86	62.71	61.77
SDM_W	$\text{SM}_{W1,1}$	1000	62.13	61.19	61.66	61.10	60.17	60.63
$\text{pDM} + \text{SDM}_W$	$\text{SM}_{1,1}$	400	65.92	70.09	67.94	63.50	67.52	65.45
$\text{pDM} + \text{SDM}_W$	$\text{SM}_{W1,1}$	2000	67.82	67.94	67.88	66.44	66.55	66.49
$\text{pDM} + \text{SDM}_W + \text{SDM}_P + \text{SM}$			72.27	70.14	**71.19**	69.23	67.19	**68.19**

zmaksymalizować F_{rec}. Widzimy, że metody wykorzystujące dodatkowo pDM uzyskują F_{rec} powyżej 68%.

Kolejnym etapem były testy metody SDM wykorzystującej do budowy słownika korpus WIKI – tę metodę nazwano SDM_W. Przetestowano tylko warianty $\text{SM}_{1,1}$ i $\text{SM}_{W1,1}$ dające najlepsze wyniki dla SDM_P. Podobnie jak dla SDM_P, przetestowano na początek samą metodę SDM – zwiększenie wielkości korpusu kilkadziesiąt razy powinno dać efekt w postaci poprawy wyników, lecz przedstawione na rys. 10.3 rezultaty są bardzo podobne do uzyskanych przez SDM_P. Ponadto dla wariantu $\text{SM}_{W1,1}$ obserwujemy spadek precyzji w stosunku do SDM_P. Następnie przetestowano metodę łączoną $\text{pDM} + \text{SDM}_W$ – wyniki przedstawiono na rys. 10.4. Wykresy dla obu wariantów są tym razem bardzo zbliżone, a wyniki



Rysunek 10.3: Wyniki testu rozpoznawania WW metodą SDM_W. Do budowania słownika (faza SM) użyto dwóch wariantów metody SM podpisanych powyżej wykresów.



Rysunek 10.4: Wyniki testu rozpoznawania WW metodą pDM + SDM$_W$. Do budowania słownika (faza SM) użyto dwóch wariantów metody SM podpisanych powyżej wykresów.

porównywalne z SDM$_P$. Druga część tabeli 10.1 podsumowuje wyniki poszczególnych wariantów SDM$_W$.

Uzyskane wyniki warte są podsumowania. Przede wszystkim obserwujemy, że mimo dużych różnic między korpusami wyniki są bardzo zbliżone – zaletą PAP-TRAIN jest podobny styl i słownictwo do próbki testowej, natomiast WIKI to z kolei dużo bogatszy zasób słownikowy. Analiza wyników w tab. 10.1 prowadzi do wniosku, że SDM$_P$ ma większą precyzję, a SDM$_W$ większą pełność, co jest zgodne z intuicją. Analizując różnice między samym SDM a pDM + SDM widzimy z kolei, jak bardzo pomaga użycie słownika o wysokiej precyzji jako metody wstępnej przed SDM. Najbardziej zastanawiające są różnice między wariantami SM$_{1,1}$ i SM$_{W1,1}$, a zwłaszcza fakt, że tan drugi wariant zyskuje więcej niż pierwszy po połączeniu z pDM – szczególnie widać to na przykładzie różnicy F_{rec} i F_{ext} dla SDM$_W$. Powodem jest prawdopodobnie fakt, że wadą SM$_{W1,1}$ jest niska precyzja, a wstępna metoda pDM może wyeliminować właśnie wyniki o niskiej precyzji, powodując, że wariant ten zyskuje dzięki wysokiej wartości wskaźnika *recall*.

10.3. Test metody łączonej

Wyniki powyższych testów skłaniają do sprawdzenia, czy udałoby się podnieść wartość F_{rec} łącząc ze sobą kilka metod. Na początek trzeba by zastosować metodę pDM – ma ona wysoką precyzję. Następnie można by użyć zarówno SDM$_W$ jak i SDM$_P$, wykorzystując jednocześnie rozmiar Wikipedii i podobieństwo notatek PAP do próbki testowej. Dodatkowo można by sprawdzić możliwość podniesienia wyników przy pomocy samej metody SM. Metoda łączona miałaby więc postać pDM+SDM$_W$+SDM$_P$+SM. Pozostaje problem doboru wariantów i parametrów metod składowych. Przeanalizujmy parametry, które trzeba będzie optymalizować:

– Dla metody SDM_W trzeba wybrać wariant SM ($\text{SM}_{1,1}$ lub $\text{SM}_{W1,1}$) oraz dobrać parametry pc_{min} i rs_{min}.

– Dla metody SDM_P parametry są takie, jak dla SDM_W.

– Dla metody SM trzeba dobrać jej wariant ($\text{SM}_{1,1}$ lub $\text{SM}_{W1,1}$) oraz wartości parametrów pc_{min} i rs_{min}.

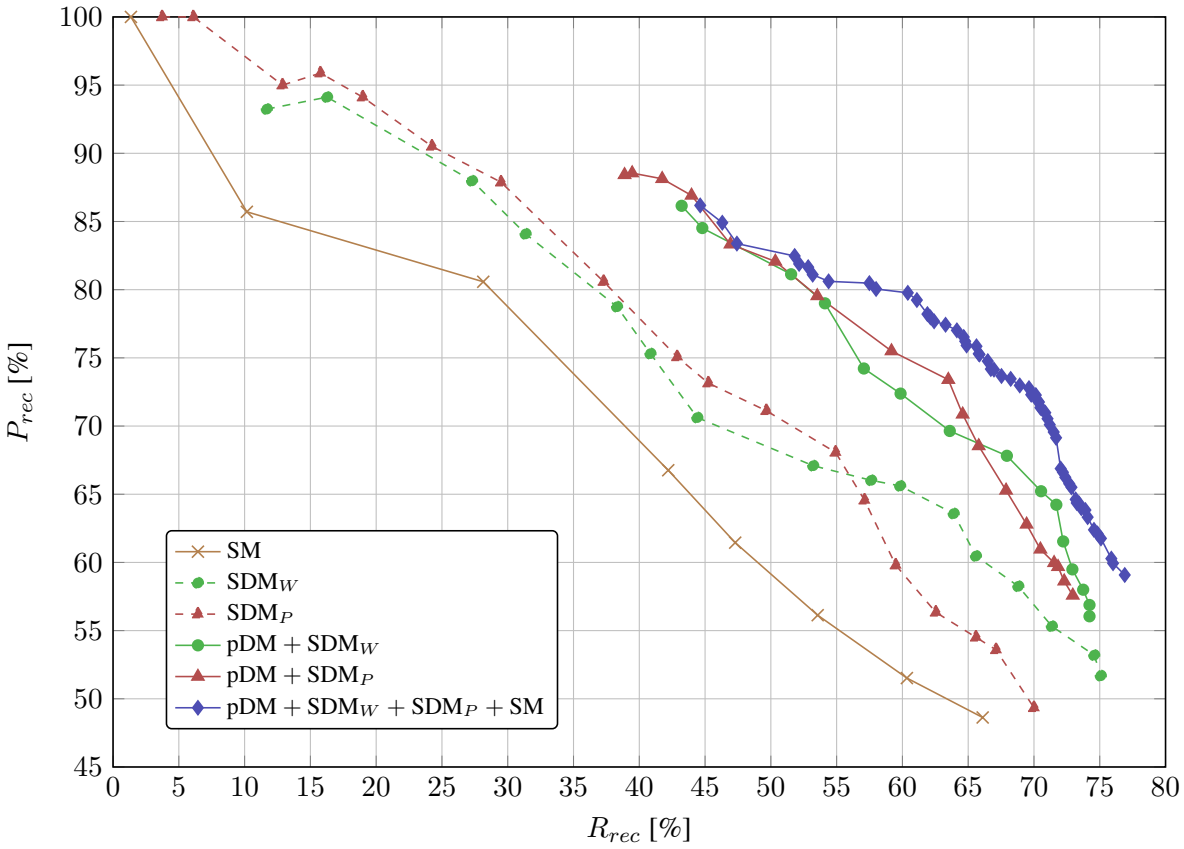
Wybór wariantu przeprowadzono analizując wszystkie kombinacje, natomiast wartości pc_{min} i rs_{min} dobrano prostą metodą *brute-force*, przeszukując wszystkie możliwości z zadaną (niewielką) dokładnością na skali logarytmicznej. Później zwiększono jeszcze dokładność, przeszukując okolice optymalnej wartości, jednak okazało się to zbędne, ponieważ metoda charakteryzuje się szerokim plateau w okolicy maksimum i niewielką zmiennością wyników, więc drobnoziarnista optymalizacja nie była konieczna. Ponownie zastosowano walidację krzyżową wyników taką, jak dla poprzednich metod (podrozdział 9.7).

Znalezione wartości optymalne odpowiadają w dużym stopniu wartościom podpowiadanym przez intuicję: dla obu metod SDM wybrano wariant $\text{SM}_{W1,1}$ – wybór ten w pewnym stopniu sugerowały już dotychczasowe wyniki, które pokazywały zalety tego wariantu w metodach łączonych. Dla metody SM wybrano z kolei wariant $\text{SM}_{1,1}$, spisujący się bezsprzecznie najlepiej również w samodzielnym teście tej metody. Wybrane wartości pc_{min} to odpowiednio 3000 dla metod SDM i 100 dla metody SM. Najciekawiej prezentują się wartości rs_{min}. Dla metody SDM_P wybrano wartość rs_{min} równą 1200, co jest niemal identyczne z dotychczasowym wyborem tego parametru (1000), natomiast okazało się, że dla SDM_W trzeba znacznie zwiększyć próg – optymalna okazała się wartość 100000, która tak naprawdę oznacza branie pod uwagę tylko niewielkiej części słownika – tej o wysokiej jakości. Z kolei dla SM wyznaczono próg $rs_{min} = 65$, który również odcina wyniki o niskiej jakości. Uzyskany wynik przedstawiono w dolnym wierszu tab. 10.1.

Warto jeszcze obejrzeć porównanie wszystkich omówionych metod na jednym diagramie – w tym celu zastosowano diagram *Precision-Recall* pokazany na rys. 10.5. Porównuje on wyniki omówionych metod, pokazując, jak dla każdej z nich wygląda kompromis między precyzją a pełnością. Dla każdej z metod przedstawiono jej najwyższe wyniki. Widzimy, że SM nie pozwala na jednoczesne uzyskanie wysokich wartości P_{rec} i R_{rec}. Metody SDM niezależnie od korpusu pozwalają tylko na niewielką poprawę. Dodanie metody pDM prowadzi jednak do znaczącej poprawy – głównie przez podniesienie precyzji. Dla metody łączonej pokazano różne punkty występujące podczas optymalizacji – wyraźnie widać, że optymalizację przeprowadzono dla wąskiego wycinka wartości P_{rec} i R_{rec} w okolicach 70% – celem była maksymalizacja F_{rec}. Przedstawiono tylko punkty nie zdominowane przez inne – inaczej zamiast łamanej otrzymalibyśmy chmurę punktów.

10.3.1. Analiza wyników testu

Metoda łączona dała zdecydowanie najlepsze wyniki testu, postanowiono więc zbadać dokładnie błędy, które popełniała. Zaimplementowane narzędzia pozwalają na dokładną analizę wyników wraz z badaniem różnic między metodami, wykorzystano je więc do analizy błędów jeden po drugim. Wybrano



Rysunek 10.5: Kompromis między wartościami współczynników *precision* i *recall* testu rozpoznawania WW dla omówionych metod ekstrakcji.

wariant dający optymalną wartość $F_{rec} = 71.19\%$. Przeanalizujmy kolejno cztery zbiory generowane podczas testu[5].

Zbiór **poprawnych wyrażeń** (T_i) zawiera **387** wyrażeń. Nie znaleziono w nim żadnych anomalii – wynik testu był prawidłowy.

Zbiór **wyrażeń z błędną etykietą** (T_d) zawiera **17** wyrażeń. Kategoria ta zawiera prawie wyłącznie imiona i nazwiska osób, których imię lub nazwisko nie znalazło się w CLPM i błędnie zostało uznane za nieodmienne, np. “Tony Blair” (3 wystąpienia), “Pete Sampras”, “Jeorga Haidera”. Jednocześnie znaleziono tutaj dwa wystąpienia: “sekretarz stanu” i “mistrzostw świata”, które w próbce testowej zostały oznaczone błędnie jako $**$ (oba segmenty odmienne) – tak naprawdę są to więc poprawne wyniki. Próbki testowej nie korygowano, ponieważ mogłoby to spowodować sztuczne odchylenie wyników w górę (*bias*).

Zbiór **wyrażeń brakujących** (F_n), czyli wyników fałszywie negatywnych zawiera **172** wyrażenia. Przyczyny braku rozpoznania są bardzo różne – zwykle bezpośrednim powodem jest długa, nietypowa struktura wyrażeń (głównie nazw), np. “V Liceum Ogólnokształcące im. Augusta Witkowskiego”, “miesięcznik "Perspektywy"”, “ustawy o wykonywaniu budżetu państwa”, “Olsztyńskich Fabryk Mebli Sp. z o.o.”, “Czwarty Zjazd Europejskiej Unii Wspólnot Polonijnych”. Ponadto pojawiają się tutaj czasami idiomy, np. “otwarcia ognia”, “czarnych chmur”, a także pojęcia i terminy, np. “kondycji gospodarczej”,

[5]Ich definicje znajdują się w podrozdziale 6.2.2.

"ministerstwo finansów", "metodą węgla C 14", "motywie zabójstwa", "rana kłuta-szarpana", "ciężarówce-chłodni" – analiza kilku takich przypadków potwierdziła, że po prostu pewne niezbyt często używane wyrażenia występujące w nietypowym konteście wymykają się zarówno metodom słownikowym (pDM, SDM), jak i metodzie SM. Ponadto czasami powodem braku rozpoznania jest błąd ortograficzny lub interpunkcyjny, np. "W.Brytania" (brak spacji po kropce), "Białego Domy". Jednocześnie kilka wyników może budzić kontrowersje co do poprawności próbki testowej, np. wyrażenia "kraje regionu", "pełnym sukcesem" i "szacunkowej wartości" co prawda są MWE, ale kwestia ich należenia do kategorii nr 2 wg listy podanej na początku rozdziału 6 może być dyskusyjna. Takich wyrazów było jedynie kilka, jednak mogą one w pewnym stopniu wpływać na wyniki, zaniżając otrzymany rezultat. Interesujący jest fakt, że wątpliwości pojawiły się dopiero podczas testu – problem ten nie został zauważony podczas tworzenia próbki testowej.

Zbiór **wyrażeń nadmiarowych** (F_p) zawiera **155** wyrażeń. Spora część z nich to wyrażenia rozpoznane częściowo, czyli mające część wspólną z jakimś wyrażeniem brakującym. Takie pary stanowią ponad jedną trzecią błędnych wyników. W wielu przypadkach są one poprawnymi WW, np. zamiast wyrażenia "Macieja N." rozpoznano "Macieja N". Czasem z kolei jeden WW został rozbity na dwa, np. zamiast "V Liceum Ogólnokształcące im. Augusta Witkowskiego" rozpoznano osobno "Liceum Ogólnokształcące" i "Augusta Witkowskiego". Ten ostatni przypadek spowodował pojawienie się 1 wyrażenia brakującego i aż 2 nadmiarowych, a tak naprawdę jest to jeden błąd. Kolejna grupa nadmiarowych wyrażeń to hasła Wikipedii, które są np. określeniami dat: "stycznia 1921", "grudniu 1981". Jest ich dość dużo i byłyby łatwe do odfiltrowania przy pomocy prostych reguł. Czasami zdarzało się tak, że rozpoznano nadmiarowo zupełnie nieprawidłowy ciąg wyrazów, np. "Umowa była", "stosunku do" – przypadki te były jednak rzadkie, co potwierdza, że metoda działa poprawnie.

Podsumowując analizę wyników możemy stwierdzić, że wynik liczbowy dość dobrze odzwierciedla rzeczywistą jakość wyników, chociaż może on być zaniżony, ponieważ większość błędów to tak naprawdę wyniki przynajmniej częściowo poprawne. Problem jednoznacznej decyzji okazał się dość trudny i pewne rzadkie przypadki sprawiają problemy zarówno metodom automatycznym, jak i człowiekowi. Zdecydowana większość wyników jest jednak jednoznaczna, a zarówno testowaną metodę jak i procedury testowe trzeba ocenić pozytywnie. Z całą pewnością konieczna jest jednak dalsza praca nad jednym i drugim – wyraźnie widać możliwości dalszej poprawy.

10.4. Analiza słownika utworzonego przy pomocy SM

Słownik utworzony podczas fazy SM algorytmu SDM jest wart dokładniejszej analizy – stanowi on wartościowy produkt, który po dodatkowej obróbce może stać się wzorcowym leksykonem WW, dołączonym do nowej wersji słownika SFJP. Postanowiono więc przetestować jego jakość. Do testów posłużył słownik tworzony z korpusu WIKI, ponieważ jest to zbiór danych dużo bogatszy od PAP-TRAIN. Ponieważ dotychczasowe testy nie wskazały jednoznacznie lepszej metody pomiędzy $\mathrm{SM}_{1,1}$ i $\mathrm{SM}_{W1,1}$, przetestowano oba warianty. Dla każdego z nich wygenerowano słownik WW z minimalnym progiem

punktowym $rs_{min} = 300$[6] – ułatwiło to testy, a ponadto poniżej tego progu jakość słowników znacznie spada. W przypadku $SM_{1,1}$ próg ten powoduje ograniczenie wielkości słownika z 5.45M[7] do 1.15M, a dla $SM_{W1,1}$ – ograniczenie z 4.03M do 1.49M. Oznaczmy zbiory słów po ograniczeniu przez D_W (dla $SM_{W1,1}$) i D (dla $SM_{1,1}$). Wyznaczono rozmiar części wspólnej: $|D_W \cap D| \approx 1.05$M. Następnie wylosowano próbkę ponad 2100 wyrazów, przy czym losowanie przeprowadzono osobno dla zbiorów $D_W \setminus D$, $D_W \cap D$ i $D \setminus D_W$, stosując rozmiary próbek proporcjonalne do wielkości tych zbiorów – ma to na celu zachowanie jednorodności rozkładu prawdopodobieństwa w sumach tych próbek. Teraz dla każdej z próbek sprawdzono ręcznie, ile jest w nich poprawnych WW, dzieląc każdą próbkę na trzy kategorie:

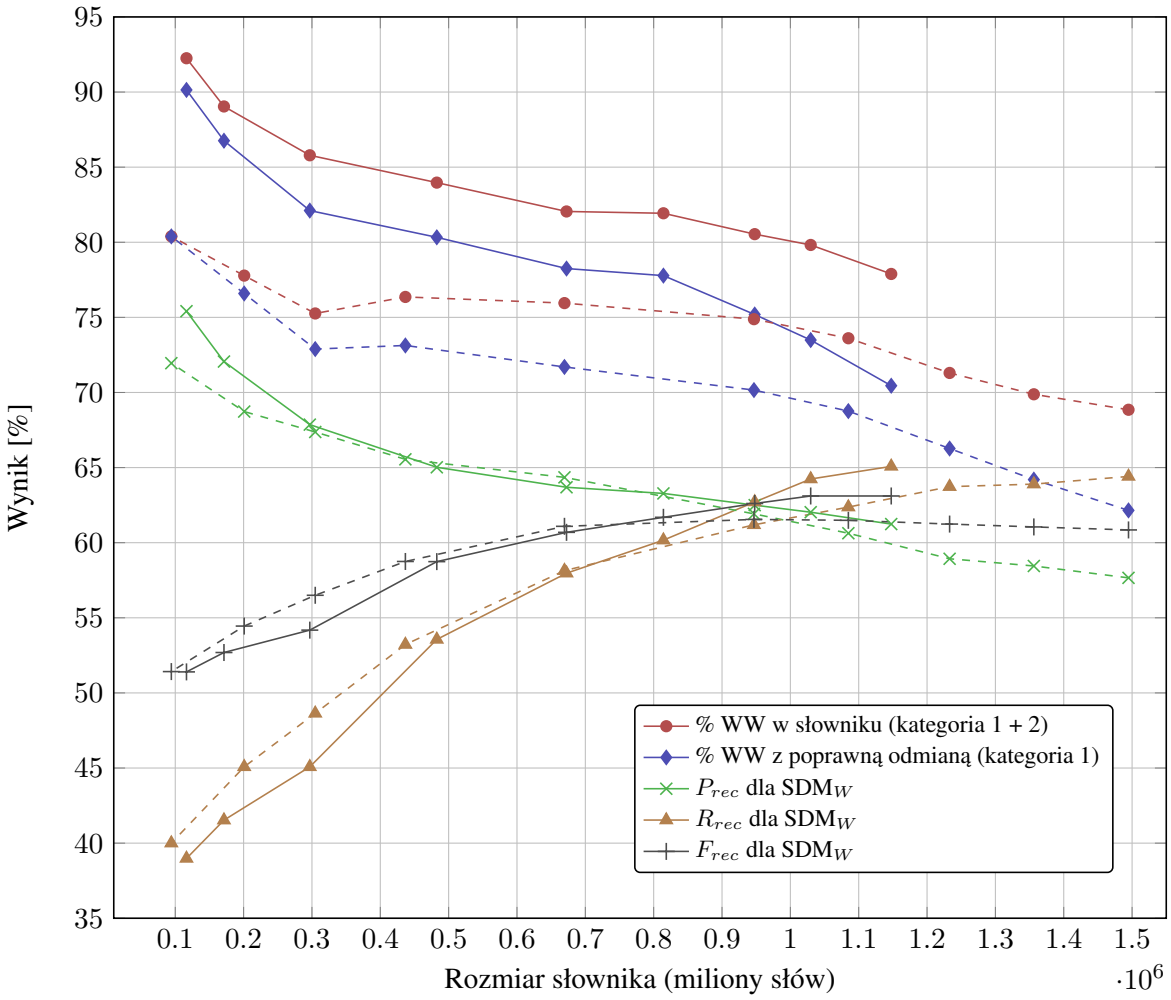
1. Wyrazy poprawne, np. “Marcin Słupski”, “rumień obrączkowaty”, “Jagnięca Ławka”, “Centralny Komitet Obywatelski”,
2. Wyrazy z błędną odmianą lub formą podstawową, np. “Ken McGregorem” oraz z błędną pisownią, np. “dolina Ważecka”,
3. Wyrażenia, które nie są WW, np. “grudzień 1795”, “pamiętny finał”, “olsztyński klub”.

Tabela 10.2: Wyniki testu jakości słownika utworzonego przez fazę SM metody SDM_W.

		Wyniki testu – % haseł w kategoriach		
Zbiór	**Wielkość zbioru [M]**	**1. poprawne**	**2. błędna odmiana**	**3. niepoprawne**
D_W	1.49	62.15	6.70	31.15
D	1.15	70.45	7.44	22.11
$D_W \cap D$	1.05	70.53	7.38	22.09
$D_W \setminus D$	0.44	43.46	5.17	51.37
$D \setminus D_W$	0.10	69.60	8.0	22.40

Okazało się, że w próbce części wspólnej wszystkie atrybuty *decl* są identyczne dla D_W i D – umożliwiło to przetestowanie części wspólnej tylko raz. Wyniki zaprezentowano w tab. 10.2. Oprócz zbiorów D_W i D przedstawiono też wynik dla części wspólnej, ponieważ wyrazy znalezione dwoma różnymi metodami powinny być bardziej wiarygodne. W rzeczywistości jednak okazuje się, że wynik części wspólnej jest praktycznie identyczny z wynikiem dla zbioru D. Można to wytłumaczyć tym, że tak naprawdę metoda $SM_{W1,1}$ dodaje pewne nowe wyrazy, więc jej wyniki w dużej części będą stanowiły nadzbiór wyników $SM_{1,1}$[8]. Rezultaty podane w tabeli dotyczą różnej wielkości słowników, więc ciężko je porównać ze sobą, a ponadto są one dość niskie (szczególnie dla $D_W \setminus D$). Można więc sprawdzić, co stanie się, gdy podniesiemy rs_{min}, zmniejszając słownik. Możemy wówczas odpowiednio przefiltrować próbki testowe – rozkład jednostajny gwarantuje dalszą poprawność wyników, chociaż ich dokładność będzie maleć – uznano, że w przypadku spadku rozmiaru próbki do ok. 100 wyrazów przerywamy zwiększanie progu. Rezultaty zaprezentowano na rys. 10.6.

[6] W przypadku słownika liczba ta oznacza minimalną wartość atrybutu *int_score*.
[7] milionów wyrazów
[8] Różnica w wielkości zbiorów początkowych na korzyść $SM_{1,1}$ wynika z zastosowanego dla $SM_{W1,1}$ progu $pc_{min} = 3000$.



Rysunek 10.6: Jakość słownika WW utworzonego przez fazę SM metody SDM$_W$ przy pomocy SM$_{1,1}$ (linia ciągła) i SM$_{W1,1}$ (linia przerywana) wraz z wynikami odpowiedniego wariantu SDM$_W$.

Oprócz wyniku testu jakości słowników na wykresie zamieszczono też dla porównania wyniki testu rozpoznawania WW dla metody SDM$_W$ – pozwala to zbadać wpływ wielkości i jakości słownika na otrzymany wynik. Różnica jakości między słownikami jest wyraźna na korzyść wariantu SM$_{1,1}$ – nie potwierdza jej jednak wynik z tab. 10.1. Dokładniejsza analiza wyników optymalnych pokazuje, że przy $rs_{min} = 300$ wielkość słownika utworzonego przez SM$_{1,1}$ wynosi 1.15M, a przy $rs_{min} = 1000$ wielkość słownika utworzonego przez SM$_{W1,1}$ to 0.95M. Dla tych wielkości różnica jakości słowników nie jest duża i znajduje odzwierciedlenie w różnicy F_{rec}. Wraz ze zmniejszaniem słownika różnica w wynikach jednak zanika, a w przedziale wielkości 0.1-0.7M przewagę (wg F_{rec}) uzyskuje wariant SM$_{W1,1}$. Wynik ten zaprzecza intuicji, więc zbadano dokładnie różnice w wynikach testu rozpoznawania WW dla SM$_{1,1}$ z progiem 5000 (oznaczmy słownik przez S) i SM$_{W1,1}$ z progiem 10000 (oznaczmy słownik przez S_W) – obie konfiguracje dają słownik zawierający ok. 0.3M wyrazów, a przy tym różnica w jakości jest tutaj największa[9]. Na rys. 10.6 dla obu metod omawiany wynik pokazuje trzeci punkt od lewej strony.

[9]Pojawia się tu też zagadkowe minimum jakości dla SM$_{W1,1}$.

Po zbadaniu różnic okazało się, że wynikają one z interesujących cech obu słowników. Słownik S pozwolił na rozpoznanie kilku nazw, których brakowało w S_W, np. "Amerykańskie Towarzystwo Psychologiczne", "Europejski Bank Inwestycyjny", "Trynidad i Tobago". Dodatkowo S nie zawiera błędnych częściowych nazw obecnych w S_W, np. "Polski Związek", "Lekka Atletyka". Sprawa wygląda jednak inaczej, gdy przyjrzymy się wyrazom pospolitym – słownik S spowodował rozpoznanie wyrażeń takich jak np. "tysiąc dolarów" (5 wystąpień), "wejście Polski", "niektóry kraj". Jednocześnie słownik ten nie zawierał poprawnych wyrazów obecnych w S_W, np. "członek rządu", "prawybory prezydenckie" – okazuje się więc, że wysoka jakość S skoncentrowała się na nazwach, które w tekście występują rzadziej niż wyrazy pospolite – a dla nich pojawiły się błędy. Dodatkowo widzimy tendencję S do zawierania większej ilości haseł Wikipedii – tłumaczy to więc różnicę na korzyść $\mathrm{SM}_{W1,1}$ w testach pDM + SDM_W – mniejsza liczba zawartych haseł Wikipedii powoduje większy zysk z użycia pDM. Charakter różnic był w rzeczywistości nieco bardziej skomplikowany, jednak omówione cechy odgrywały rolę dominującą.

Powyższa analiza wyraźnie pokazuje, że wyrażona procentowo jakość słownika niekoniecznie przekłada się wprost na jakość ekstrakcji z jego udziałem. Jest tak dlatego, ponieważ w teście istotne jest, aby wyrażenia często występujące w próbce testowej były rozpoznane poprawnie. A zatem zarówno brak w słowniku poprawnych, ale rzadko występujących w tekście wyrazów, jak i obecność w słowniku słów zupełnie niepoprawnych, ale bardzo rzadko występujących w tekście, nie ma większego wpływu na wynik. Dlatego też procentowa jakość słownika niekoniecznie musi być bardzo wysoka – ważne, by zawierał on wszystkie często występujące WW. Pojawia się pytanie – jak dużo wyrazów powinien zawierać słownik WW? Autor pracy [13] próbuje odpowiedzieć na to pytanie bazując na oszacowaniach liczby MWE pamiętanych przez człowieka. Okazuje się jednak, że różni cytowani autorzy uzyskiwali wartości od kilku tysięcy do kilku milionów. Pokazuje to ponownie, jak trudno jest ocenić rzeczywistą wartość słownika WW znając tylko jego wielkość i procentową jakość. Mimo wszystko wydaje się, że uzyskana jakość sięgająca 75-80% dla słownika o wielkości 1M pozwala wykorzystać SM do tworzenia dobrego zasobu słownikowego zawierającego WW – słownik ten wymagał jednak będzie dodatkowego przetwarzania. Proponowane ulepszenia przedyskutowano w podrozdziale 11.1.

11. Możliwości dalszego rozwoju

W poprzednich rozdziałach opisano algorytmy ekstrakcji wyrazów wielosegmentowych z tekstu, pozwalające na osiągnięcie dobrych wyników na korpusie testowym ($F_{rec} > 71\%$). Wszystkie te metody służą tak naprawdę jednemu celowi – stworzeniu zestawu narzędzi umożliwiających przetwarzanie WW dla języka polskiego. Metody te pozwalają więc w pewnym stopniu zapełnić lukę po prawej stronie tabeli 1.1 przedstawionej we wstępie. Mimo uzyskania dobrych wyników trzeba jednak przyznać, że jest to jedynie pierwszy etap prac prowadzących do tego, by dla języka polskiego istniały zasoby zawierające WW o rozmiarach i jakości pozwalającej na swobodne używanie ich do przetwarzania tekstu. Zastanówmy się więc, jakie powinny być kolejne kroki prowadzące do tego celu.

11.1. Dalszy postęp ekstrakcji WW

Podczas pracy nad kolejnymi metodami ekstrakcji WW zauważono stały, powolny wzrost wartości F_1 dla testów rozpoznawania i ekstrakcji – przedstawione jako rezultat wartości osiągnięto w wyniku stosowania kolejnych niewielkich ulepszeń. Dlatego też proces ten można kontynuować, uzyskując dalsze postępy. W przypadku metod DM i pDM wykorzystujących słownik utworzony z haseł Wikipedii istnieje możliwość poprawy ich wyników poprzez użycie manualnie tworzonych **reguł**. Tabela 7.4 pokazuje, że nawet prosta reguła heurystyczna przedstawiona na rys. 7.6 pozwoliła w dużym stopniu podnieść skuteczność metody DM. Można więc podjąć próbę stosowania bardziej rozbudowanych reguł do tworzenia wzorców odmiany dla tych haseł Wikipedii, dla których wzorzec odmiany nie mógł zostać wyznaczony w sposób automatyczny.

Metodą, która wydaje się dawać największe szanse poprawy, jest SM[1] – wynika to z faktu, że tak naprawdę jest to tylko jedna z wielu możliwych metod poszukiwania wyrazów o cechach zbliżonych do haseł Wikipedii. Struktura wzorców syntaktycznych jest dopracowana i wielokrotnie testowane były różne jej warianty, lecz również tutaj możliwa jest dalsza poprawa. W szczególności wydaje się, że powinna istnieć możliwość wykorzystania wzorców, w których niektóre segmenty odmienne są już ustalone – byłoby to więc połączenie wzorców słownikowych i syntaktycznych. Przykładowo, segment “Związek” często rozpoczyna nazwy różnych organizacji. Tego rodzaju wzorce można by wyekstrahować w oparciu o dane statystyczne występowania najczęstszych segmentów w pewnym słowniku WW (samej Wikipedii lub np. słowniku utworzonym metodą SM). Poza tym istnieje też możliwość użycia metod prezentowanych w

[1] Rozumiana również jako faza SM metody SDM.

innych pracach dotyczących MWE i NER celem poprawy wyników metody SM – przeanalizujmy je po kolei.

Rozważmy na początek możliwości wykorzystania **uczenia bez nadzoru**, ponieważ jedno z założeń niniejszej pracy polegało na tym, że nie stosowano żadnych ręcznie tworzonych zbiorów treningowych. W podrozdziale 2.2 omówiono już istniejące metody, których wyniki są raczej niskie – przykładem mogą być tutaj opisywane już miary MI, χ^2, PI i EPI, które osiągają niską precyzję nawet dla prostego testu klasyfikacji binarnej (tak/nie) [77]. W takim razie pozostaje użycie **uczenia z nadzorem**. Klasyfikatory, np. SVM bazujące na miarach statystycznych wymienionych powyżej nie dają dobrych wyników [78], natomiast potraktowanie elementów wzorców syntaktycznych jako cech służących do wytrenowania klasyfikatora na próbce pozytywnych i negatywnych przykładów MWE wydaje się dawać większe szanse powodzenia [20]. Najlepszym wyborem wydaje się zastosowanie tagera np. opartego o CRF [14, 15], który trzeba wytrenować na korpusie otagowanym MWE. Ponieważ nie są znane odpowiednio duże korpusy dla j. polskiego otagowane MWE, pozostaje możliwość ponownego wykorzystania Wikipedii: można potraktować linki do haseł Wikipedii występujące w korpusie WIKI jako tagi, podobnie jak w pracy [62] – jeżeli dla jakiegoś hasła wyekstrahowano wzorzec odmiany, linki przychodzące powinny być wysokiej jakości. Można również rozważyć dołączenie wystąpień danego hasła w całym korpusie WIKI. Tager wytrenowany na takiej próbce ma szanse osiągnąć wysoką precyzję, przy czym ze względu na obecność nieotagowanych WW[2] w korpusie treningowym wartość współczynnika *recall* będzie niska. Dołączenie takiego tagera jako nowego składnika metody łączonej może jednak wyraźnie podnieść jej skuteczność.

Kolejną grupą narzędzi wspomagających ekstrakcję MWE i NER są mechanizmy oparte na **regułach**, np. Spejd [8]. Użycie tego narzędzia wprost byłoby trudne ze względu na odmienność formatu danych, jednak mogłoby ono okazać się użyteczne zwłaszcza w kontekście wykrywania grup składniowych – pozwoliłoby to np. odrzucać potencjalne MWE, które nie są wyrażeniami rzeczownikowymi oraz ujednoznaczniać segmenty WW[3]. Warto również przeanalizować możliwość użycia istniejących reguł dla narzędzia SProUT [66, 65, 89] służących do NER.

Same wzorce syntaktyczne można również traktować jako automatycznie utworzone reguły, jednak z pewnością można wzbogacić je o pewne reguły tworzone ręcznie. W szczególności należy podkreślić, że wzorce syntaktyczne w obecnej formie bardzo często powodują uznanie za WW wyrazów, które nimi nie są. Jest to spowodowane faktem, że nie ma mechanizmu filtrującego przypadki fałszywie pozytywne poza odcięciem wyników na danym poziomie liczby wystąpień. Można więc, podobnie jak to uczyniono dla algorytmu ekstrakcji etykiet semantycznych (rozdział 5), dodać ręcznie utworzone reguły filtrujące wyniki fałszywie pozytywne, zmniejszając tym samym konieczność użycia progu jakości słów rs_{min} w słowniku utworzonym metodą SM. Reguły takie można utworzyć w oparciu o statystykę wystąpień haseł pogrupowanych wg wspólnych segmentów. W ten sposób zobaczymy, że np. hasła pasujące do wyrażenia regularnego "lipiec [0-9]+" nie są WW. W końcu możliwa jest też ręczna (a właściwie

[2] Metody pDM i DM osiągały pełność w okolicach 40%, co sugeruje, że tylko taka część WW jest hasłami Wikipedii.

[3] Ostateczny efekt jest jednak niepewny, ponieważ w przeciwieństwie do metod wykorzystywanych w niniejszej pracy parsing powoduje wybór tylko jednej spośród wielu możliwych struktur składniowych – nie zawsze poprawnej.

półautomatyczna) poprawa najczęściej występujących słów tego słownika, prowadząca do utworzenia zasobu słownikowego, który potencjalnie może być używany do wytrenowania klasyfikatorów, np SVM, pozwalających na odfiltrowanie słów o niskiej jakości z pozostałej części słownika. W szczególności algorytm SM mógłby posłużyć za metodę ekstrakcji kandydatów, a klasyfikator umożliwiałby wybór kandydatów o wysokiej jakości, zastępując prostą metodę progu odcięcia rs_{min}. Oczywiście dobór odpowiednich cech wykorzystywanych w klasyfikacji jest złożonym problemem.

Kolejną możliwością poprawy jest użycie dodatkowych zasobów: **korpusów** i **słowników**. Przede wszystkim w teście SDM okazało się, że pewne znaczenie ma wybór korpusu, na którym wykonywana jest faza SM. Można więc użyć dodatkowych korpusów – jedną z możliwości jest np. ostatnio dostępny wśród zasobów Katedry Informatyki Akademii Górniczo-Hutniczej korpus notatek prasowych PAP z okresu 30 lat. Korpus NKJP otagowany NE [71] może z kolei posłużyć do wzbogacenia słownika metody DM/pDM. Ponadto zasób ten można by wykorzystać podobnie jak autorzy korpusu do treningu tagera, który poprzez wykrywanie NE podniósłby wyniki ekstrakcji MWE. Korpusu otagowanego NE można też użyć do fazy SM, tworząc więcej wzorców syntaktycznych na podstawie wystąpień NE. Ponadto dla języka polskiego dostępny jest ostatnio leksykon zawierający ponad milion nazw wielosegmentowych, który cały czas jest rozwijany [46]. Również SłowoSieć [49] zawiera kilka tysięcy WW. Mogą one więc w zauważalnym stopniu pozwolić na podniesienie wyników SDM. WordNet zawiera ponadto informacje z zakresu **semantyki**, które mogłyby zostać wykorzystane jako dodatkowe dane do ekstrakcji WW. O ile dołączenie klasy semantycznej poszczególnych segmentów do wzorców syntaktycznych prawdopodobnie spowodowałoby zbyt duży spadek wskaźnika *recall*[4], to np. omówiony powyżej klasyfikator filtrujący wyrazy słownika mógłby wykorzystywać tę informację. Dodatkowo można rozważyć rozpoznawanie wzorem NKJP [71] wyliczeń typu “premierzy Polski i Czech”, “z województw: małopolskiego i podkarpackiego”. Wymagałoby to rozszerzenia istniejących automatów lub być może wykorzystania częściowego parsingu jako etapu wstępnego.

Badano również możliwość zastosowania **metod iteracyjnych**, polegających na wykorzystaniu słownika tworzonego przy użyciu SM do tworzenia większej liczby wzorców syntaktycznych, które w kolejnej iteracji służyłyby do ponownej rozbudowy słownika. Okazuje się jednak, że każda kolejna iteracja traci precyzję dużo bardziej, niż zyskuje pełność. Jednak niewykluczone, że po poprawie słownika tworzonego przy pomocy SM będzie on miał na tyle wysoką jakość, by mógł zostać wykorzystany do utworzenia większej ilości wzorców syntaktycznych.

Na koniec warto zauważyć, że duża część problemów z ekstrakcją WW dla języka polskiego znikłaby, gdyby istniał duży **korpus otagowany WW** – można by wówczas wykorzystać tager oparty np. o CRF, który wraz ze słownikiem WW mógłby pozwolić na osiągnięcie wartości F_1 nawet powyżej 80%[5]. Warto więc rozważyć możliwość półautomatycznego tagowania korpusu za pomocą opracowanych w niniejszej pracy metod. Ponadto tworząc otagowany korpus można by poprawić możliwości testowania algorytmów

[4] Wzorce stałyby się zbyt rzadkie i niektóre nie wystąpiłyby ani raz – oczywiście warto to zbadać.
[5] Oszacowanie to opieramy na fakcie, że taką skuteczność osiągano w pracach [14, 15]

ekstrakcji WW – korpus testowy mógłby być wówczas dużo większy. Głównym problemem jest tutaj fakt, że wynikowy korpus musi być duży, by trening tagera był efektywny.

Podsumowując, podjęcie wymienionych powyżej kroków powinno doprowadzić do utworzenia kompletnego **słownika wyrazów wielosegmentowych** dla języka polskiego, ulepszonego **algorytmu ekstrakcji WW** oraz **korpusu otagowanego WW**. Narzędzia te powinny umożliwić przetwarzanie WW na poziomie syntaktycznym z dużą dokładnością.

11.2. Klasyfikacja semantyczna wyrazów

We wstępie do niniejszej pracy (rozdział 1) przedstawiono tabelę podsumowującą zasoby leksykalne dla języka polskiego (tab. 1.1). Z zestawienia tego wynika, że oprócz samej ekstrakcji WW potrzebne są metody ich klasyfikacji semantycznej. Pierwszym etapem takiej klasyfikacji są **etykiety semantyczne** opisane w rozdziale 5. Są one wyznaczane jednak wyłącznie dla haseł Wikipedii. Istnieje więc potrzeba ekstrakcji takich etykiet dla dowolnych wyrazów wielosegmentowych wyekstrahowanych z tekstu. Jest to problem złożony, ponieważ w tekście nie znajdziemy bezpośredniej informacji na temat znaczenia danego wyrazu.

11.2.1. Wzorce syntaktyczne a etykiety semantyczne

Na początku rozdziału 9 podano przykład sugerujący, że powinien istnieć związek miedzy wzorcami syntaktycznymi a etykietami semantycznymi[6]. Postanowiono więc wykonać eksperyment badający możliwość wyznaczenia etykiety semantycznej ekstrahowanych WW na podstawie tych wzorców. W pierwszej kolejności należy wykorzystać utworzone już etykiety semantyczne dla haseł Wikipedii (wierszy tabeli `WikiEntry`). Wzorce odmiany i wzorce syntaktyczne wyznaczane są dla wierszy tabeli `Word`, więc można przypisać im etykiety odpowiednich haseł Wikipedii. Pierwszy problem polega na tym, że niektóre hasła (ok. 10%) nie posiadają etykiet semantycznych – możemy je po prostu pominąć. Ponadto kilku hasłom (tabela `WikiEntry`) może odpowiadać jeden wyraz z wzorcem odmiany (tabela `Word`)[7]. Zdecydowano więc, że do wyznaczania etykiet brane będą pod uwagę tylko te wyrazy, dla których istnieje dokładnie jedna etykieta, czyli np. będzie tak dla kilku wsi o nazwie “Nowa Biała” z jedną etykietą “wieś”, ale już nie dla nazwy “Czarna Góra”, mogacej oznaczać m.in. wieś lub szczyt. Dodatkowo w niniejszym eksperymencie bierzemy pod uwagę jedynie rzeczownik główny etykiety, czyli np. dla “rodzaj ssaka” będzie to “ssak”, a dla “włoski kolarz szosowy” – “kolarz”.

Przydzielenie etykiet wyrazom wielosegmentowym z wzorcami daje bezpośrednio jedynie możliwość wyznaczenia etykiety dla wyrazów rozpoznanych przy pomocy metody pDM. Celem jest jednak przydzielenie etykiety wyrazom rozpoznanym przy pomocy SM. Zdecydowano, że etykiety będziemy przyznawać nie wyrazom rozpoznanym w tekście, lecz wyrazom będącym elementami słownika WW utworzonego przy pomocy SM (posiadającym atrybut *is_final* = `True`), a ponadto stosujemy wysoki próg

[6] Przykład dotyczył m.in. związków chemicznych: “tlenek węgla”, “siarczan miedzi”, “wodorotlenek sodu” itp.

[7] Jest tak, gdy kilka wierszy tabeli `WikiEntry` ma taki sam atrybut *name* (podrozdział 8.1).

rs_{min}. Ma to zapewnić odpowiednią liczbę wystąpień wzorców syntaktycznych, zwiększając szansę na poprawne wyznaczenie etykiety. Pozostaje więc zdefiniowanie algorytmu wyznaczającego etykietę. Zauważmy, że skoro udało się przydzielić etykiety wierszom tabeli `Word`, oznacza to również jednoznaczne ich przypisanie wierszom tabeli `ContextPatternBinding` (patrz rys. 9.3). Przypomnijmy teraz, że ocena wyniku wyrazu (tabela `WordResult`) jest sumą ocen odpowiednich wyników wzorca (tabela `ContextPatternResult`), a z kolei ocena wyniku wzorca wyliczana jest na podstawie statystyki wzorca kontekstu (atrybut *forms* tabeli `ContextPattern`). Ponieważ w tabeli `ContextPatternBinding` przechowywane są również statystyki wzorca (także w atrybucie *forms*), ale dla konkretnego wyrazu źródłowego, można wyznaczyć ocenę wyniku wzorca oddzielnie dla każdego wyrazu źródłowego, a co za tym idzie, dla każdej etykiety. Docelowo w każdym wyniku wzorca otrzymamy statystykę w postaci listy par (l, N), gdzie l to etykieta, a N – liczba wystąpień. Zsumowanie tych statystyk na poziomie wyniku wyrazu (`WordResult`), a następnie na poziomie wynikowego wyrazu (`Word`) w słowniku WW pozwala otrzymać taką statystykę dla każdego wyrazu. Pozostaje teraz wybrać etykietę z najwyższą liczbą wystąpień, przy czym wartościowa może być także analiza etykiet na dalszych miejscach. Ważne jest, by zdać sobie sprawę, że drobnoziarnistość etykiet oraz brak bezpośredniej implikacji miedzy syntaktyką a semantyką powodują, że wynik tego eksperymentu musi być niski – jest to jedynie proste doświadczenie.

Nie wyznaczano procentowej skuteczności, ponieważ jest ona niska – wstępne testy pokazały dokładność poniżej 25% dla wyrazów ze słownika liczącego jedynie 171 tys. wyrazów uzyskanego dla $s_{min} = 10000$. Test przeprowadzono dla kilkudziesięciu losowo wybranych wyrazów słownika nie posiadających jeszcze etykiet semantycznych. Kilka przykładowych rezultatów pokazano w tab. 11.1. Poprawne etykiety uzyskiwane są tylko dla kategorii typowych dla danej struktury syntaktycznej. Na przykładzie trzech pierwszych wyników widzimy, jak fakt obecności wielu nazw gmin, powiatów, prowincji itp. w Wikipedii ma wpływ na przydzielane etykiety. Kolejne rezultaty pokazują, że co prawda imiona i nazwiska są rozpoznawane poprawnie, jednak kategorie, do których należy dana osoba, są za każdym razem zbliżone i dość przypadkowe. Należy tu jednak podkreślić, że zadanie znalezienia poprawnej etykiety jest czasem trudne nawet dla człowieka, ponieważ zwykle istnieje wiele osób o danym imieniu i nazwisku. Ostatni przykład pokazuje, jak nazwa mająca strukturę typową dla uczelni spowodowała przydzielenie właśnie takiej etykiety.

Tabela 11.1: Wyniki automatycznego wyznaczania etykiet semantycznych dla kilku przykładowych haseł słownika WW. Pogrubiono poprawne rezultaty.

Wyraz	**Poprawna etykieta**	**Wynik 1**		**Wynik 2**		**Wynik 3**	
		l	N	l	N	l	N
gmina Cabanelles	gmina	**gmina**	21138	prowincja	16910	prefektura	4115
polana Butorzonka	polana	gmina	12681	prowincja	10146	prefektura	2469
sport szybowcowy	sport	powiat	290727	jednostka	15710	obwód	11482
Włodzimierz Skoczylas	aktor	**aktor**	5505	polityk	5003	poeta	4922
Maciej Gliszczyński	piłkarz	aktor	2944	poeta	2843	polityk	2510
Dworek Cisowy	dworek	uczelnia	7574	uniwersytet	4757	region	4045

Warto wspomnieć, że przetestowano też wersję powyższego algorytmu pobierającą wspólne **hiperonimy** wyznaczanych etykiet z WordNetu – niestety wyniki nie poprawiły się znacznie. Oczywiście dla osób otrzymano poprawną etykietę "człowiek", lecz w innych przypadkach nie zaobserwowano poprawy. Pokazuje to ostatecznie, że ograniczona informacja syntaktyczna to zbyt mało dla poprawnego wyznaczenia etykiety semantycznej.

Tabela 11.2: Najczęściej występujące rzeczowniki główne etykiet semantycznych haseł Wikipedii.

Etykieta	Liczba
wieś	80765
miejscowość	61051
miasto	42180
piłkarz	39707
planetoida	30375
gmina	27375
polityk	18318
album	17314
klub	16089
film	12938
stacja	12405
parafia	10479
osada	10449
ryba	9619
zespół	9285
aktor	8853
lekkoatleta	8225
gromada	8033
duchowny	7683
związek	7242
galaktyka	6501
aktorka	6495
roślina	6059
ważka	5937
jednostka	5904
lekkoatletka	5792
przystanek	5563
ptak	5492
kościół	5482
rzeka	5368
szczyt	5364
ryby	5341

11.2.2. Przyszłe prace nad etykietami semantycznymi

Rozważmy teraz możliwości rozwoju algorytmów przydzielających etykiety semantyczne. Z całą pewnością konieczne jest użycie **uczenia z nadzorem**, czyli algorytmu, który użyje treningowego zbioru WW oznaczonych etykietami semantycznymi do tego, by przydzielać takie etykiety nowym wyrazom. Algorytmy klasyfikacji mają jednak ważne ograniczenie – jest nim liczba klas. Niestety liczba różnych etykiet semantycznych przydzielonych obecnym hasłom Wikipedii to ponad 160 tys. Ponadto analizując etykiety dostrzegamy ich znaczną drobnoziarnistość. We wstępie do niniejszej pracy stwierdzono, że etykiety semantyczne powinny same również znaleźć się w słowniku – powinna być dostępna informacja na temat ich odmiany (słownik CLPM lub słownik WW) i znaczenia (słownik semantyczny lub ontologia). W takim razie w pierwszej kolejności dalsze prace powinny zmierzać w kierunku sprowadzenia etykiet do mniejszego zbioru **klas semantycznych**, dla których znaczenie będzie dobrze znane. Ponadto warto rozważyć wykorzystywanie, wzorem innych autorów (np. [94, 42]), kategorii i infoboksów z Wikipedii do poprawy dokładności etykiet semantycznych.

Po analizie statystyki wystąpień poszczególnych etykiet przydzielonych hasłom Wikipedii okazało się, że gdybyśmy zostawili tylko 5000 najczęściej występujących rzeczowników głównych etykiet, to oznaczonych etykietami pozostałoby 89% haseł. Najczęściej występujące rzeczowniki główne pokazano w tab. 11.2. Oczywiście nie zawsze wystarczy sam rzeczownik główny – niektóre etykiety są WW, co można rozpoznać przy pomocy utworzonego w niniejszej pracy słownika WW (i ewentualnie WordNetu) – dla nich najprawdopodobniej trzeba będzie pozostawić dodatkowe elementy etykiety. Po wybraniu i uproszczeniu najczęściej występujących etykiet powinno być możliwe sprowadzenie ich w sposób półautomatyczny do znacznie mniejszej liczby klas – wstępne oszacowania sugerują, że około 50 klas semantycznych wystarczy do tego, by dość dokładnie skategoryzować słowa. Oczywiście nie ma jednej poprawnej odpowiedzi

– dlatego też ontologie i WordNet posiadają złożone hierarchie bytów. Z drugiej strony struktury te są czasem zbyt skomplikowane i abstrkacyjne, co utrudnia ich zastosowanie w praktyce [54, 18].

Po wyborze docelowego zbioru klas kolejnym zagadnieniem jest dobór **zbioru treningowego**. Naturalny wydaje się ponowny wybór Wikipedii z hasłami oznaczonymi klasami semantycznymi, są jednak również inne możliwości – można wykorzystać zasoby takie jak WordNet, NELexicon czy też ontologia CYC, przy czym wymaga to mapowania klas występujących w tych zasobach do zbioru klas wyznaczonego powyżej.

Mając dany zbiór treningowy, trzeba przeprowadzić dobór odpowiednich **cech**. Mogą one obejmować cechy syntaktyczne i semantyczne zarówno samego wyrazu (i jego poszczególnych segmentów), jak i kontekstu. W tym celu można posłużyć się istniejącymi publikacjami[8]. Należy też zwrócić uwagę na fakt, że etykieta niektórych haseł często jest zawarta w samym haśle (słowa “gmina”, “polana”, “sport”, “dworek” z tab. 11.1) – warto to uwzględnić dokonując selekcji cech. Być może trzeba będzie też wprowadzić ręczne **reguły**.

Jedną z najważniejszych decyzji jest również wybór **algorytmu** klasyfikującego. W obecnej chwili wydaje się, że najlepsze wyniki uzyskiwane są przy użyciu klasyfikatora SVM, chociaż warto również przetestować np. model maksymalnej entropii oraz tagowanie segmentów klasami semantycznymi przy uzyciu tagerów wykorzystujących np. model CRF. Ciężko jest przewidzieć wyniki takiej klasyfikacji – wydaje się jednak, że rezultat powinien być pozytywny, biorąc pod uwagę istniejące prace poświęcone klasyfikacji NE [98]. Z całą pewnością istnieje konieczność prowadzenia dalszych badań w tej dziedzinie, jednak zdecydowano się nie obejmować ich już zakresem niniejszej pracy.

[8] Analiza stanu badań w tym zakresie nie została jeszcze przeprowadzona.

12. Podsumowanie

W niniejszej pracy przedstawiono metody ekstrakcji wyrazów wielosegmentowych z tekstu w języku polskim, a dodatkowo omówiono algorytm wyznaczania etykiet semantycznych dla haseł Wikipedii. Nadrzędnym celem tworzenia i rozwoju tych metod i narzędzi jest uzupełnienie zasobów leksykalnych umożliwiających przetwarzanie WW w tekście polskim.

We wstępie do pracy postawiono dwie tezy (podrozdział 1.1). Przeprowadzone badania wykazują ich prawdziwość. **Teza 1** została udowodniona przez wyniki uzyskane przez metody SM, SDM oraz metodę łączoną. Zaprezentowane rezultaty pokazują, że istnieje możliwość automatycznej ekstrakcji wyrazów wielosegmentowych z tekstu przy pomocy słownika fleksyjnego i artykułów Wikipedii bez wykorzystania dodatkowych reguł i zbiorów treningowych. Metoda łączona przedstawiona w rozdziale 10.3 uzyskała w teście rozpoznawania wyrazów wielosegmentowych wartość F_1 przekraczającą 71%, a w teście ekstrakcji – 68%, co pozwala stwierdzić, że teza ta została potwierdzona. Prawdziwość **Tezy 2** wykazują z kolei przedstawione metody tworzenia słownika WW z Wikipedii (metody DM i pDM) i z wyników działania algorytmu SM (analiza tego słownika znajduje się w podrozdziale 10.4).

12.1. Efekty pracy

Podstawowym efektem pracy są zaprojektowane, zaimplementowane i przetestowane mechanizmy do ekstrakcji wyrazów wielosegmentowych z tekstu w języku polskim. Najważniejsze algorytmy zaprojektowane i zaimplementowane w pracy to:

- Algorytm wyznaczający wzorce odmiany haseł Wikipedii.
- Algorytmy tworzenia wzorców słownikowych i syntaktycznych dla haseł Wikipedii.
- Algorytm tworzący automat rozpoznający wzorce (słownikowe lub syntaktyczne).
- Algorytm ekstrahujący wyrazy wielosegmentowe korzystając z utworzonego automatu.
- Metody ekstrakcji wyrazów wielosegmentowych przy pomocy słownika: DM i pDM.
- Metoda ekstrakcji wzorców wielosegmentowych przy pomocy wzorców syntaktycznych: SM.
- Metoda SDM oraz metoda łączona ekstrakcji wyrazów wielosegmentowych.
- Algorytm ekstrakcji etykiet semantycznych dla haseł Wikipedii[1].

[1] Główna część opracowana w pracy magisterskiej [10].

Algorytmy te pozwoliły na zaimplementowanie systemu, w skład którego wchodzą m.in. poniższe narzędzia:

- Narzędzie tworzące słownik CLPM na podstawie danych z CLP i Morfologika.
- Narzędzie ekstrahujące dane z Wikipedii i Wikisłownika.
- System ekstrahujący etykiety semantyczne z haseł Wikipedii.
- Narzędzie pozwalające na ekstrakcję wyrazów wielosegmentowych różnymi metodami oraz tworzące słownik WW z wyniku ekstrakcji.
- Korpus testowy PAP-WW oraz mechanizm testowania algorytmów ekstrakcji WW przy jego pomocy.

Narzędzia te z kolei pozwalają utworzyć zasoby takie jak:

- Słownik CLPM.
- Baza danych utworzona z Wikipedii przechowująca m.in. strony, artykuły, linki i kategorie.
- Korpus WIKI.
- Słownik WW utworzony z haseł Wikipedii z wzorcami odmiany.
- Słownik WW utworzony przy pomocy metody SM.
- Automaty rozpoznające wyrazy wielosegmentowe przy pomocy różnych metod.
- Baza danych zawierająca wyniki testów poszczególnych metod.

Istotną właściwością pracy jest możliwość ponownego wygenerowania wszystkich powyższych zasobów na podstawie dowolnej wersji Wikipedii w ciągu 1-2 dni. Całość dostępna jest jako jeden system zaimplementowany w języku Python – istnieje więc możliwość samodzielnej instalacji i utworzenia wszystkich zasobów opisanych w niniejszej pracy na podstawie polskiej Wikipedii. Poniżej opisano krótko najważniejsze narzędzia wchodzące w skład zaimplementowanego systemu.

12.1.1. Narzędzia pomocnicze

Pierwszym narzędziem, które zostało utworzone, jest **słownik CLPM** opisany w rozdziale 3.4. Łączy on w sobie dane słownika CLP i Morfologika – jest to jedyne narzędzie dla języka polskiego łączące te zasoby. Ponadto charakteryzuje go unikalny format danych, który z jednej strony wykorzystuje ideę etykiet fleksyjnych i numerów form znanych z CLP, a z drugiej strony pozwala na zapis oboczności. Kolejną zaletą jest zwięzłość znaczników słownikowych w porównaniu ze znacznikami morfosyntaktycznymi używanymi w Morfeuszu i Morfologiku. Warto też podkreślić dużą wydajność użytej bazy danych LMDB, która została wysoce zoptymalizowana, dzięki czemu szybkość działania tego słownika dla typowych operacji przewyższa bibliotekę CLP[2]. Z pewnością jest to wartościowe narzędzie i **zostanie ono wykorzystane** przy tworzeniu nowej generacji słownika SFJP.

[2]Testy zostały wykonane dla języka Python. W języku C przewaga CLPM mogłaby zaniknąć.

Kolejnym narzędziem wchodzącym w skład utworzonego systemu jest **mechanizm ekstrakcji danych z Wikipedii** opisany w rozdziale 4. Pozwala on na łatwe przetworzenie aktualnej wersji Wikipedii do bazy danych SQLite. Gdy rozpoczynano prace, nie istniał jeszcze zasób, który mógłby zastąpić to narzędzie. Obecnie sytuacja się zmieniła, ponieważ istnieją projekty takie, jak np. zawierająca m.in. treści infoboksów powiązane w ontologię DBPedia [42], posiadająca również dane dla języka polskiego. Mimo to jednak sposób ekstrakcji artykułów, prowadzący do utworzenia korpusu WIKI, bazy linków z kontekstami, ekstrakcji pierwszego akapitu artykułów itp. jest dalej unikalny dla niniejszej pracy – pozwala on przeprowadzić dalsze etapy badań. Ponadto warto podkreślić, że omawiany zasób może być generowany w dowolnej chwili z najnowszej wersji Wikipedii w czasie 2-3 godzin[3]. Trzeba również wspomnieć, że równolegle powstało narzędzie ekstrahujące dane z Wikisłownika.

12.1.2. Narzędzia do ekstrakcji wyrazów wielosegmentowych

Główny zestaw narzędzi służy do ekstrakcji wyrazów wielosegmentowych. Pierwszym z nich jest mechanizm tworzenia słownika wyrazów wielosegmentowych z haseł Wikipedii, wyznaczający **wzorce odmiany** dla poszczególnych haseł. Jest to mechanizm, który może mieć szerokie zastosowania, zastępując lub uzupełniając często używane tworzone manualnie reguły odmiany, które nie zawsze są poprawne. Warto pokreślić, że mechanizm ten może działać **nie tylko dla Wikipedii** – może on zostać użyty dla dowolnego zestawu WW, które występują w pewnym korpusie tekstów. Jego przyszłe zastosowania mogą być szerokie.

Następnym elementem utworzonego pakietu jest **algorytm tworzenia wzorców syntaktycznych** z haseł Wikipedii. W niniejszej pracy jest on używany jako podstawa metody SM (a więc i SDM), natomiast w przyszłości może on zostać wykorzystany do **generowania cech** dla klasyfikatora używanego w metodach uczenia z nadzorem (np. SVM, MaxEnt). Utworzone wzorce są zoptymalizowane jednocześnie pod kątem zwięzłości, szybkości przetwarzania, jak i czytelności dla człowieka – w porównaniu z alternatywnymi formatami są dużo krótsze i łatwiejsze w prezentacji, ponieważ zostały zoptymalizowane pod konkretne zastosowanie.

Kolejny utworzony mechanizm to zoptymalizowany **automat rozpoznający wzorce** (podrozdział 7.3), który może zostać użyty zarówno do rozpoznawania WW, jak i wzorców syntaktycznych. Co prawda istnieją już narzędzia regułowe takie jak Spejd [8], jednak unikalną cechą utworzonego automatu jest optymalizacja pod kątem równoczesnego rozpoznawania w tekście nawet milionów wzorców wraz z jednoczesną ekstrakcją cech syntaktycznych. Wysoką wydajność zapewniają dwie implementacje (`DeepMap` i `MDBMap`) zoptymalizowane odpowiednio czasowo i pamięciowo. System oferuje możliwość zrównoleglania rozpoznawania wzorców.

Z powyższym automatem powiązany jest system przechowywania **wyników ekstrakcji**, który umieszcza wyekstrahowane wyrazy w bazie danych, pozwalając na konstrukcję **słownika wyrazów wielosegmentowych**. Przetwarzanie korpusu WIKI zawierającego ponad 202M segmentów wraz z zapisem wyników trwa w zależności od rodzaju wzorców od 2 (DM) do kilkunastu godzin (SM). Użyty format zapisu

[3] Prace prowadzono na maszynie z procesorem Intel Core i7-4770K, 16 GB RAM i dyskiem SSD.

danych pozwala złożoną **analizę wyników** oraz realizację **metod iteracyjnych**, polegających na użyciu utworzonego słownika lub wzorców syntaktycznych w kolejnej fazie przetwarzania oraz **łączenie metod ekstrakcji**, które pozwala zwiększać jakość wyników. Wszystkie metody mogą być zrówoleglane przez podanie parametru określającego liczbę procesów.

Kolejnym efektem pracy jest **korpus testowy** PAP-WW zawierający 100 notatek prasowych PAP otagowanych wyrazami wielosegmentowymi. Pozwala on na testowanie i porównywanie metod ekstrakcji WW. Oczywiście ma on ograniczony rozmiar, a użyta definicja WW jest dość specyficzna, jednak jest to pierwszy krok do utworzenia wysokiej jakości korpusów (testowych i treningowych) otagowanych WW. Użyty format może zostać wykorzystany w innych pracach, chociaż ma on pewne ograniczenia.

12.1.3. Narzędzie ekstrahujące etykiety semantyczne

Kolejne utworzone narzędzie (opisane w rozdziale 5) pozwala na **ekstrakcję etykiet semantycznych** z utworzonej powyższą metodą bazy danych. Wykorzystywany jest algorytm heurystyczny, umożliwiający otrzymanie bazy, w której dla niemal każdego hasła Wikipedii dotyczącego jakiegoś rzeczywistego obiektu będzie się znajdowała etykieta semantyczna, pozwalająca na odczytanie jego znaczenia. Dokładność etykiet przekracza 94%, a istnieją możliwości dalszej poprawy (opisane w podrozdziale 11.2.2). Algorytm ten jest pierwszym takim narzędziem znanym Autorowi dla języka polskiego – pierwsza wersja powstała już jako główna część pracy magisterskiej [10].

12.2. Zakończenie

Przetwarzanie tekstu w języku polskim jest zagadnieniem trudnym zarówno ze względu na cechy języka, jak i na jego niedużą popularność, co przekłada się na brak odpowiednich narzędzi i zasobów leksykalnych. Szczególnym problemem sa wyrazy wielosegmentowe, które chociaż łączą w sobie kilka wyrazów znanych w danym języku, to jednak posiadają własną strukturę i znaczenie. Powoduje to konieczność tworzenia słowników, korpusów i sieci semantycznych zawierających te wyrazy – tylko w ten sposób można umożliwić ich przetwarzanie w tekście. Wynika stąd potrzeba rozwoju narzędzi i algorytmów umożliwiających tworzenie takich zasobów poprzez ekstrakcję wyrazów wielosegmentowych z tekstu. W niniejszej pracy przedstawiono pierwsze[4] narzędzie dla języka polskiego ekstrahujące z tekstu wszystkie wyrazy wielosegmentowe pełniące rolę rzeczownikową. Utworzone algorytmy i narzędzia pozwalają na tworzenie zasobów językowych zawierających wyrazy wielosegmentowe dla języka polskiego – efekt pracy jest więc bardzo znaczący. Dodać należy, że w dziedzinie przetwarzania wyrazów wielosegmentowych dla języka polskiego obserwujemy w ostatnich kilku latach wyraźny postęp i jeżeli tylko uda się połączyć wysiłek różnych zespołów badawczych, może się okazać, że narzędzia tworzone dla naszego języka ojczystego dorównają, a może nawet przewyższą jakością te dla języków takich jak angielski czy francuski. Wizja przyszłego rozwoju przetwarzania języka polskiego jest więc pozytywna.

[4] znane Autorowi pracy

13. Dodatek

Przegląd funkcjonalności projektu

Projekt został zaimplementowany w języku Python w wersji 2.7 [73]. Dane przechowywane są bazie danych SQLite [93], natomiast za odwzorowanie obiektowo-relacyjne odpowiada framework Django [17]. Dane, dla których czas odczytu ma znaczenie krytyczne (słownik CLPM, automaty rozpoznające wzorce), przechowywane są w bazie danych LMDB [43]. Projekt posiada w pełni konfigurowalną strukturę katalogów, która domyślnie wygląda następująco:

```
.
├── data ..... dane projektu
│   ├── corp ..... korpusy
│   ├── db ..... pliki baz danych SQLite
│   ├── fix ..... reguły poprawiające etykiety semantyczne
│   ├── mdb ..... pliki baz danych LMDB
│   ├── morf ..... dane Morfologika i pomocnicze dane słownika CLPM
│   ├── test ..... pomocnicze pliki testowe i wyniki testów
│   └── wiki ..... dane pobrane z Wikipedii
├── log ..... logi
├── nlp ..... źródła projektu
│   └── wikidict ..... źródła i testy głównej aplikacji
└── run.py ..... skrypt startowy
```

Podstawowy sposób używania projektu polega na uruchomieniu skryptu startowego poleceniem:

```
$ python -i run.py
```

Korzystanie z projektu sprowadza się wówczas do wywoływania odpowiednich funkcji w interpreterze Pythona w sposób interaktywny. Innym sposobem jest użycie skryptu startowego do wykonania pewnych poleceń w trybie wsadowym – może on przyjmować argumenty oznaczające operacje do wykonania, więc przykładowo poniższa komenda spowoduje pobranie danych z Wikipedii i utworzenie etykiet semantycznych:

```
$ python run.py getdump df
```

Do wykonywania operacji takich jak powyższe warto użyć systemu CI (*continuous integration*), ponieważ mogą one trwać długo – w projekcie użyto narzędzia Jenkins [31].

W niniejszym rozdziale zademonstrowane zostaną kluczowe elementy funkcjonalności projektu. Opis projektu nie jest kompletnym podręcznikiem użytkownika, a jedynie przeglądem jego podstawowych możliwości. Wszelkie przykładowe polecenia będą wykonywane interaktywnie w interpreterze Pythona uruchomionym przez skrypt startowy.

13.1. Tworzenie słownika CLPM

Aby korzystać z projektu, w pierwszej kolejności należy przygotować słownik CLPM. W tym celu konieczne jest zainstalowanie biblioteki CLP [26]. Dodatkowo aby dane słownikowe były kompletne, należy też zainstalować narzędzie Morfologik[1] [57, 56]. W celu wyekstrahowania danych z CLP konieczne jest umieszczenie listy numerów ID poszczególnych wyrazów w pliku `data/morf/dict.clp`[2]. Dane Morfologika powinny zostać umieszczone w pliku o analogicznej nazwie `data/morf/dict.morf`. Dane te należy zapisać w formacie TSV (*tab-separated value*). Plik ten możemy utworzyć automatycznie:

```
>>> morf.extract_to_file()
```

Następnie tworzymy bazę danych słownika. W obecnej wersji konieczne są dwa przebiegi – pierwszy pobiera informacje o przypadkach szczególnych, czyli słowach, które ze słownika muszą zostać wykluczone. Są to np. krótkie rzeczowniki takie jak “do”, “in”. Drugi przebieg zapisuje słownik bez tych wyrazów. Podczas tworzenia słownika (oraz pozostałych operacji opisywanych w tym rozdziale) można obserwować postępy i obsługę różnych przypadków wyjątkowych w logach systemowych[3].

```
>>> clpm.prebuild_mdb()
>>> clpm.build_mdb()
```

Należy teraz ponownie uruchomić skrypt startowy, ponieważ do tej pory korzystał on ze zwykłego słownika CLP. Dopiero teraz będzie on wykorzystywał CLPM. W katalogu `data/mdb` powinny powstać pliki bazy danych słownika.

[1] Użycie Morfologika jest opcjonalne – cały projekt będzie działał poprawnie bez niego, chociaż wyniki będą niższe.

[2] Nazwa pliku `dict` wynika z faktu, że domyślnie wynikowy słownik nazywa się *dict*. Możliwe jest też utworzenie więcej niż jednej wersji słownika pod różnymi nazwami i późniejsze przełączanie między nimi.

[3] Projekt współpracuje z linuksowym systemem logowania `syslog-ng` [96].

```
$ ls -lh data/mdb/dict*.mdb
-rwxrwxr-x 1 pch users 119M 03-09 15:15 data/mdb/dict-form.mdb
-rwxrwxr-x 1 pch users 247M 03-09 15:15 data/mdb/dict-rec.mdb
```

Plik `dict-form.mdb` zawiera tabelę `Form`, a plik `dict-rec.mdb` – tabelę `Rec`[4]. Implementacja słownika znajduje się w module `morf`, w klasie `MDBDict`. Obiekt słownika posiada atrybuty `fd` (obiekt klasy `FormDB`) i `rd` (obiekt klasy `RecDB`) – umożliwiają one dostęp do bazy LMDB. Format zapisu danych jest maksymalnie zoptymalizowany pod względem czasu odczytu, a przy tym poszczególne formy tekstowe i ich numery zakodowane są na minimalnej liczbie bitów w postaci jednego łańcucha znaków Unicode – każdy numer formy zapisany jest jako jeden znak Unicode, a poszczególne formy oddzielane są bajtem zerowym. Okazało się, że taki format daje najmniejsze czasy odczytu, znacznie niższe niż np. w przypadku użycia modułu `struct`[5] czy `pickle`[6], a przy tym jest bardzo efektywny pamięciowo. Dodatkowo celem podniesienia szybkości działania dla głównych metod słownika użyto LRU cache [82]. Sam dostęp do słownika zapewnia moduł `clpm` – podczas inicjalizacji sprawdza on, czy baza LMDB jest dostępna i w zależności od tego w zmiennej `clp` umieszcza słownik CLPM (obiekt klasy `morf.MDBDict`) lub CLP (obiekt klasy `clp.CLP`, czyli interfejs do biblioteki CLP napisanej w C). Wykonajmy kilka przykładowych operacji korzystając ze słownika CLPM.

```
>>> len(clpm.clp.fd)  # liczba form podstawowych (rozmiar tabeli Form)
414250
>>> len(clpm.clp.rd)  # liczba form tekstowych (rozmiar tabeli Rec)
3943687

# rozpoznajmy możliwe identyfikatory dla tekstu 'woli'
>>> clpm.clp.rec(u'woli')
(u'CC-woli', u'ADA-wola', u'B-woleć', u'AEA-wole')

# identyfikator zawiera etykietę fleksyjną i formę podstawową
>>> clpm.clp.label(u'ADA-wola'), clpm.clp.bform(u'ADA-wola')
(u'ADA', u'wola')

# jakim numerom form rzeczownika 'wola' odpowiada tekst 'woli'?
>>> clpm.clp.vec(u'ADA-wola', u'woli')
(2, 3, 6, 9)

# nowa operacja 'tag' łączy w sobie 'rec' i 'vec'
>>> clpm.clp.tag(u'woli')     # otrzymujemy znacznik słownikowy
{u'ADA-wola': [2, 3, 6, 9],
 u'AEA-wole': [9],
```

[4] Opis tych tabel znajduje się w podrozdziale 3.4.
[5] `https://docs.python.org/2/library/struct.html`
[6] `https://docs.python.org/2/library/pickle.html?highlight=cpickle#module-cPickle`

```
 u'B-woleć': [4],
 u'CC-woli': [1, 7, 8, 11, 14, 29, 35]}

# możemy też w łatwy sposób otrzymać tekstowy znacznik słownikowy
>>> clpm.strtag(u'woli')
u'ADA-wola:2,3,6,9|AEA-wole:9|B-woleć:4|CC-woli:1,7,8,11,14,29,35'

# wektor form został zastąpiony przez słownik, co pozwala na oboczności
>>> clpm.clp.formd(u'ACA-pal')
{u'palem': [5], u'pale': [8, 11, 14], u'palami': [12], u'palowi': [3],
 u'palach': [13], u'palu': [6, 7],
 u'pali': [9], u'palów': [73],     # formy alternatywne D l.mn. (9 + 64 = 73)
 u'pala': [2], u'pal': [1, 4], u'palom': [10]}
```

13.2. Ekstrakcja danych z Wikipedii

Pobieranie i ekstrakcję danych z zasobów Wikipedii zaimplementowano w module `getdump`. Cała operacja wykonywana jest automatycznie. W pierwszej kolejności należy utworzyć bazę danych SQLite, podając wybraną przez siebie nazwę bazy. System może pracować z wieloma bazami danych, więc zalecane jest przechowywanie kolejnych stanów bazy w osobnych plikach, co pozwala łatwo wrócić do poprzedniego stanu[7]. Użyty w projekcie system Jenkins dodatkowo ułatwia takie podejście. Utwórzmy więc bazę danych *raw*, zawierającą wyłacznie dane pobrane z Wikipedii.

```
>>> db.new('raw')     # powstanie plik bazy danych SQLite 'data/db/raw.db'
>>> getdump.create()
```

Funkcja `getdump.create` wykona w sposób automatyczny poszczególne kroki ekstrakcji. Możemy je też wykonywać pojedynczo:

```
# pobierz aktualny zrzut danych Wikipedii do katalogu 'data/wiki'
>>> getdump.prepare()

# wyekstrahuj strony specjalne (szablony i kategorie) Wikipedii i Wikisłownika
>>> getdump.extract_spec(WIKI)
>>> getdump.extract_spec(WIKT)

# wyekstrahuj strony Wikipedii (z treścią) i Wikisłownika
>>> getdump.extract(worker_num=7, source=WIKI, with_content=True)
>>> getdump.extract(worker_num=7, source=WIKT)
```

[7] Alternatywnie możemy wszystkie dane przechowywać w jednej bazie, jeżeli zasoby dyskowe są ograniczone.

Zrzut pobierany jest tylko wówczas, gdy pojawiły się nowe, jeszcze nie pobrane dane. Ekstrakcja stron wykonywana jest w sposób równoległy – wykorzystywany jest model, w którym proces dostarczający dane (`DataProvider`) udostępnia kolejkę zadań, a poszczególni wykonawcy (`Worker`) pobierają kolejne zadania z kolejki, po czym wysyłają rezultat do procesu zapisującego wyniki do bazy (`DBWriter`). Argument `worker_num` określa liczbę wykonawców. Liczba 7 okazała się optymalna dla procesora o 4 rdzeniach i 8 wątkach sprzętowych[8]. Sam proces ekstrakcji i zapisywania danych omówiony został w podrozdziałach 4.1 i 4.2. Gdy ekstrakcja się zakończy, możemy zacząć korzystać z utworzonej bazy. Poniżej pokazano kilka przykładowych operacji na wyekstrahowanych danych.

```
# znajdź strony o nazwie 'Nowa Biała'
>>> WikiEntry.objects.filter(name=u'Nowa Biała')
[<WikiEntry: Nowa Biała>,
 <WikiEntry: Nowa Biała (województwo mazowieckie)>,
 <WikiEntry: Nowa Biała (województwo małopolskie)>]

# wybierz miejscowość z woj. małopolskiego
>>> we = WikiEntry.objects.get(title=u'Nowa Biała (województwo małopolskie)')

# domyślnie wyświetlany jest pierwszy akapit artykułu
>>> we.article
<Article: Nowa Biała (słow. Nová Belá, węg. Újbéla) - wieś w Polsce położona w
↪  województwie małopolskim, w powiecie nowotarskim, w gminie Nowy Targ.>

# szablony artykułu
>>> we.article.special_pages.filter(ns=10)
[<SpecialPage: Szablon:Wieś infobox (10)>,
 <SpecialPage: Szablon:Przypisy (10)>,
 <SpecialPage: Szablon:Dopracować (10)>,
 <SpecialPage: Szablon:Gmina Nowy Targ (10)>,
 <SpecialPage: Szablon:Osobny artykuł (10)>]

# kategorie artykułu
>>> we.article.special_pages.filter(ns=14)
[<SpecialPage: Kategoria:Nowy Targ (gmina wiejska) (14)>,
 <SpecialPage: Kategoria:Spisz (14)>]

>>> we.article.out_links.count() # liczba linków wychodzących z artykułu
41
>>> we.in_links.count() # liczba linków przychodzących do hasła
23
```

[8] Jest to logiczne biorąc pod uwagę obecność procesów `DataProvider` i `DBWriter`, które wykorzystują niemal w pełni jeden wątek.

13.3. Tworzenie etykiet semantycznych

Za tworzenie etykiet semantycznych odpowiada moduł `df`[9], natomiast moduł `dfdb` odpowiada za operacje bazodanowe. Pojedynczą etykietę dla podanego hasła można utworzyć przy pomocy funkcji `dfdb.create_definition`. Podstawowy sposób użycia polega jednak na utworzeniu etykiet dla wszystkich haseł Wikipedii w sposób automatyczny – służy do tego metoda `dfdb.defdict.create`. Jeżeli chcemy zachować stan surowej bazy danych przed utworzeniem etykiet, możemy skopiować bazę do nowego pliku – nazwijmy go *labels*:

```
>>> db.cp('labels')        # skopiuj bazę do nowego pliku 'labels'
>>> db.set('labels')       # używaj nowej bazy
>>> dfdb.defdict.create()  # utwórz etykiety dla haseł Wikipedii
```

Po utworzeniu etykiet semantycznych możemy sprawdzić rezultaty:

```
>>> WikiEntry.objects.exclude(df=None).count() # liczba haseł z etykietami
1301953
>>> WikiEntry.objects.filter(df=None).count()  # liczba haseł bez etykiet
128626

>>> we=WikiEntry.objects.get(title=u'Kraków')
>>> we.df
<Df: miasto>

>>> we=WikiEntry.objects.get(title=u'Kwas siarkowy')
>>> we.df
<Df: nieorganiczny związek chemiczny>
>>> we.df.main_index    # pozycja rzeczownika głównego licząc od 0
1

>>> we=WikiEntry.objects.get(title=u'Kot domowy')
>>> we.df
<Df: gatunek ssaka>
>>> we.df.word          # etykieta z operatorami
<Word: gatunek ssaka>
>>> we.df.simple_word   # etykieta bez operatorów
<Word: ssak>
```

[9]Etykiety są w kodzie nazywane definicjami w celu uniknięcia niejednoznaczności z etykietami fleksyjnymi, stąd skrót *df*.

13.4. Ekstrakcja WW metodą DM

Główną funkcjonalnością projektu jest ekstrakcja WW. Na początek przedstawiona zostanie najprostsza metoda DM (patrz rozdział 7). Bazuje ona bezpośrednio na hasłach Wikipedii, więc baza *raw* utworzona w podrozdziale 13.2 jest wystarczająca. Użyjemy jednak bazy *labels* zawierającej etykiety semantyczne, ponieważ chcielibyśmy przetestować również wariant DM, który bierze pod uwagę tylko hasła z etykietami.

W pierwszej kolejności należy utworzyć automat rozpoznający wzorce. Istnieje możliwość bezpośredniego utworzenia automatu dla podanych haseł. W tym przypadku można wykorzystać wariant `DeepMap`, który implementuje automat w postaci zagnieżdżonych słowników (patrz podrozdział 7.3). Automat konstruujemy, tworząc obiekt klasy `dpm.DeepMap`[10]. Utwórzmy więc automat odpowiadający przykładowi z podrozdziału 7.3.

```
>>> dmap = dpm.DeepMap([u'Park Narodowy Glacier',
                        u'Park Decjusza',
                        u'Park Narodowy Glacier Bay'])
>>> dmap.count_dicts()  # liczba użytych słowników (obiektów dict)
9

>>> for cp in dmap.cps: print cp.p  # wzorce słownikowe
...
"Park","park",WACA-park:1,wACA-park:1 "Narodowy",WCC-narodowy:1,WCC-narodowy:8
↪  "Glacier"
WACA-park:1,wACA-park:1 "Decjusza"
"Park","park",WACA-park:1,wACA-park:1 "Narodowy",WCC-narodowy:1,WCC-narodowy:8
↪  "Glacier" "Bay"
```

Powyższy automat rozpoznaje tylko trzy wzorce. Podstawowy sposób użycia polega jednak na utworzeniu automatu rozpoznającego wszystkie hasła Wikipedii. W tym wypadku zaleca się używanie wariantu `MDBMap`, który zapisuje automat do bazy danych LMDB. Pozwala to utworzyć automat tylko raz, dzięki czemu kilkuminutowa operacja nie musi być powtarzana po ponownym uruchomieniu systemu. Pozwala to również znacząco zmniejszyć zużycie pamięci.

```
# upewniamy się, że pracujemy na bazie 'labels'
>>> db.cur()
('default', 'labels')

# utwórz automat o nazwie 'dm' dla wszystkich haseł Wikipedii
>>> dpm.MDBMap('dm', patdb.gen_wiki_words())
```

[10]Nazwa modułu jest skrótem od *dictionary pattern matching*.

W porównaniu z `DeepMap` konstruując obiekt klasy `MDBMap` podajemy dodatkowo jako pierwszy argument nazwę tworzonej bazy. W katalogu `data/mdb` powinny pojawić się pliki bazy danych automatu:

```
$ ls -lh data/mdb/pmutil*.mdb
-rwxrwxr-x 1 pch users 416M 03-09 16:28 data/mdb/pmutil-cps-dm.mdb
-rwxrwxr-x 1 pch users 166M 03-09 16:28 data/mdb/pmutil-map-dm.mdb
```

Przedrostek `pmutil` w nazwach plików to nazwa modułu, który zawiera definicje odpowiednich klas odpowiadających za zapis i odczyt bazy. Plik `pmutil-cps-dm.mdb` zawiera listę wzorców słownikowych, natomiast plik `pmutil-map-dm.mdb` zawiera opis kolejnych przejść automatu – dla każdego przejścia $s \xrightarrow{p} t$ kluczem w tabeli jest para (s, p), a wartością – nowy stan t.

Tworzenie automatu jest na tyle typową operacją, że w module `patdb` zdefiniowano funkcje pomocnicze, które upraszczają tę operację. Utwórzmy więc automaty odpowiadające kolejnym wariantom metody DM przedstawionym w tabeli z wynikami 7.4.

```
# automat 'dm' identyczny jak poprzednio
>>> patdb.create_we_dmap('dm')

# automat 'dm-df' tylko dla wyrazów z etykietą semantyczną
>>> patdb.create_we_dmap('dm-df', require_df=True)

# automat 'dm-mse' dla heurystycznie wyznaczonych wzorców odmiany
>>> patdb.create_mse_dmap('dm-mse')
```

Wykorzystajmy teraz automat *dm* do ekstrakcji WW z przykładowego zdania. Do przeprowadzenia ekstrakcji służy klasa `MultiMatcher` – implementuje ona algorytm 4 z podrozdziału 7.4.2. Posiada ona metodę `match`, która wymaga podania obiektu klasy `CorpusEntry`, zawierającego tekst, dla którego przeprowadzana jest ekstrakcja. Obiekt ten może zostać odczytany z bazy danych lub utworzony ręcznie. W poniższym przykładzie utworzono go ręcznie – pozwala to na przetestowanie ekstrakcji bez modyfikacji bazy danych. Wyniki również nie są zapisywane do bazy.

```
>>> ce = CorpusEntry(text=u'W Parku Narodowym Glacier Bay zamieszkują zające
↪  amerykańskie')
>>> pmap = dpm.MDBMap('dm')        # wczytaj automat 'dm' z bazy LMDB
>>> mm = dpm.MultiMatcher(pmap)    # utwórz obiekt wykonujący ekstrakcję
>>> mm.match(ce)                   # wykonaj ekstrakcję
>>> info(mm.results)               # wypisz wyniki ekstrakcji
-------------------------------------------------------------------------------
1 W Parku Narodowym Glacier Bay
2         1 Parku Narodowym Glacier Bay
3             1 Park Narodowy Glacier Bay ACA-park:1 CC-narodowy:1,8   6
4         1 Parku Narodowym Glacier
```

```
5              1 Park Narodowy Glacier ACA-park:1 CC-narodowy:1,8  6
6   zamieszkują zające amerykańskie
7          1 zające amerykańskie
8              1 zając amerykański ABA-zając:1 CC-amerykański:1,8 8,11,14
```

Format wypisywania wyników jest wart omówienia. Podświetlone fragmenty oznaczają sekcje (obiekty `Section` – patrz podrozdział 7.5.1). Dla każdej sekcji wypisywana jest statystyka wyników sekcji (obiekty `SectionResult`). Wynik wybrany (dla którego *pos* = 0) wyświetlany jest na zielono, a pozostałe na czerwono. Przed każdym wynikiem wypisywana jest ocena danego wyniku (dla metody DM wynosi ona 1). Widzimy więc, że podobnie jak w przykładzie w podrozdziale 7.5.2 wybrano dłuższy wynik "Parku Narodowym Glacier Bay". Dla każdego z wyników sekcji pokazywana jest z kolei statystyka wyników wyrazu (`WordResult`). Tutaj również może wystąpić wiele pozycji[11], a ocena wybranego wyniku jest oznaczona kolorem zielonym.

Format wyniku wyrazu przeanalizujemy na przykładzie linii 3. Na początku wypisywane są segmenty wyrazu – odmienne na niebiesko, a nieodmienne na czarno. W omawianym przypadku dwa pierwsze segmenty są odmienne, a dwa kolejne nieodmienne. Następnie znajdziemy zapisaną na różowo listę znaczników słownikowych (atrybut *tags*) poszczególnych segmentów. W opisywanym przykładzie na końcu znajdują się dodatkowe spacje oznaczające dwa segmenty nieodmienne. Ostatnie pole to lista rozpoznanych numerów form WW, zapisana kolorem jasnoniebieskim. W tym przypadku hasło rozpoznano w formie miejscownika l.poj. (forma nr 6).

Jeżeli chcemy przeprowadzić rozpoznawanie na większym korpusie tekstu, wówczas najlepiej użyć funkcji `patdb.dmatch`, która pozwala na zrównoleglenie całego procesu i zapis wyników do bazy. Podczas zapisu wykorzystywany jest dodatkowy atrybut *iteration* – dotyczy on wszystkich tabel związanych z wynikami i pozwala na jednoczesny zapis w tej samej bazie wyników kliku różnych operacji ekstrakcji. Domyślnie każda nowa operacja ekstrakcji otrzymuje numer iteracji o 1 większy od poprzedniej. Dla przykładu pokażemy, w jaki sposób można przeprowadzić ekstrakcję metodą DM na kompletnym korpusie notatek prasowych PAP. Korpus ten można przekonwertować do postaci bazy danych poniższą funkcją – musi on być jednak dostępny w katalogu `data/corp`.

```
>>> ptxt.create_pap_corpus()
>>> CorpusEntry.objects.filter(corpus='pap').count()  # liczba notatek prasowych
51573

>>> db.cp('dm-test'); db.set('dm-test')  # nowa baza danych dla testu
>>> pm.get_iteration(Word)  # pobierz obecny numer iteracji
0
```

[11] W omawianym przykładzie występuje zawsze jeden wynik – nie jest to regułą.

```
# operacja ekstrakcji, może potrwać kilka minut - postęp pokazują logi
>>> patdb.dmatch('dm', corpus='pap')

>>> pm.get_iteration(Word)  # zmienił się numer iteracji
1

# wypisz wyniki ekstrakcji dla przykładowej notatki nr 20202
>>> pmi.corpus_entry_info('#020202')
```

```
1  Marszałek Sejmu
2          1 Marszałek Sejmu
3                  1 Marszałek Sejmu AAA-Marszałek:1|AAA-marszałek:1 ACA-sejm:2 1
4                  1 marszałek Sejmu AAA-marszałek:1 ACA-sejm:2 1
5   Maciej Płażyński
6          1 Maciej Płażyński
7                  1 Maciej Płażyński AAA-Maciej:1 AAA-Płażyński:1,7 1
8                  1 Maciej Płażyński AAA-Maciej:1 AAA-Płażyński:1 1,7
9                  1 Maciej Płażyński AAA-Maciej:1 AAA-Płażyński:1 1
10  zakończył trzydniową, nieoficjalną wizytę w Rosji. (...)
```

Przykład ten wyraźnie pokazuje główną wadę metody DM – brak wiedzy na temat odmienności segmentów powoduje brak możliwości wyboru poprawnego wzorca odmiany dla wyrazu “Maciej Płażyński”. Z tego powodu wybrany zostaje w sposób losowy zły wariant, w którym segment “Płażyński” jest nieodmienny (linia 7).

13.5. Ekstrakcja WW metodą pDM

Pierwszym krokiem do ekstrakcji WW metodą pDM jest wyznaczenie w sposób automatyczny wzorców odmiany dla haseł Wikipedii (podrozdział 8.1). Na początku należy podjąć decyzję, czy podczas tworzenia wzorców mamy uwzględnić wystąpienia hasła w bieżącym artykule (patrz podrozdział 8.2) lub nawet całej Wikipedii (eksperyment opisany w podrozdziale 8.3). Rozważmy przypadek, w którym chcemy uwzględniać wystapienia w całej Wikipedii. Wówczas w pierwszej kolejności musimy utworzyć korpus WIKI, a następnie dokonać w nim rozpoznawania haseł Wikipedii specjalnym wariantem metody DM. Służy do tego funkcja `patdb.pdmatch` – poniżej pokazano jej użycie. Bazą źródłową może być *raw* lub *labels*. Przed wykonaniem całej operacji dane kopiowane są do nowej bazy *pdwiki*.

```
>>> db.cp('pdwiki'); db.set('pdwiki')

>>> ptxt.create_wiki_corpus()
>>> CorpusEntry.objects.filter(corpus='wiki').count()
1019268  # liczba stron w korpusie 'wiki', pomijane są strony bez tekstu
```

```
>>> patdb.pdmatch('dm')  # domyślnie corpus = 'wiki'
# operacja ta może trwać kilka godzin
```

Do tworzenia wzorca odmiany dla pojedynczego hasła służy moduł `pat`. Za tworzenie wzorca odpowiada klasa `LinkPattern`. Przykładowo, aby utworzyć wzorzec odmiany dla hasła "Wydra kanadyjska", należy wpisać:

```
>>> lp = pat.LinkPattern(u'Wydra kanadyjska')
```

W pierwszej kolejności przeanalizujmy linki przychodzące dla tego hasła:

```
>>> lp.link_info()
1                            wydra kanadyjska
2                            wydra kanadyjska
3                            wydra kanadyjska
4                            wydra kanadyjska
5                            wydra kanadyjska
6  ...i, bóbr kanadyjski oraz wydra kanadyjska.
7  ...: bobra kanadyjskiego i wydrę kanadyjską; te zwierzęta, podobnie...
8  ...la, bobra kanadyjsiego, wydrę kanadyjską oraz nielicznie występu...
9  ...owcę kanadyjską, bobra, wydrę kanadyjską, wiewiórkę, pręgowca am...
10 ...ncerniki, szopy pracze, wydry kanadyjskie, jelenie wirginijskie i...
11                           Lontra canadensis
12 ...rzęta łowne takie, jak: wydra kanadyjskie, kuny amerykańskie, zaj...
```

Kolorem niebieskim oznaczono linki zgodne z wyznaczonym wzorcem odmiany, a kolorem czerwonym – wystąpienia niezgodne z wzorcem. Widzimy, że połowa linków nie posiada kontekstu – wynika to z faktu, że nie są to wystąpienia hasła w tekście, lecz w różnych zestawieniach, listach, tabelach itp. Wyświetlmy teraz dokładniejsze informacje o wyznaczaniu samego wzorca odmiany – pokazane też są informacje na temat wyznaczonych wzorców syntaktycznych, ponieważ klasa `LinkPattern` wyznacza również te wzorce.

```
>>> info(lp)
1 wydra kanadyjska *ad1 *cc15
2     6 wydra kanadyjska [1] (1)
3             g0         oraz wydra kanadyjska.     _p
4     3 wydrę kanadyjską [4] (3)
5              p            , wydrę kanadyjską,     _p
6             g0            i wydrę kanadyjską;     _p
7              p            , wydrę kanadyjską oraz  g
```

```
 8    2 1 wydry kanadyjskie [8, 11, 14] (2)
 9                p            , wydry kanadyjskie,    _p
10               g0            i wydry kanadyjskie (    p
11    1 0 Wydra kanadyjska [1] (1)
12                 |             Wydra kanadyjska (     p
13    1 0 wydry kanadyjskiej [2] (1)
14              ac9  podgatunków wydry kanadyjskiej:   _p
15      1 wydra kanadyjskie (1)
16      1 Lontra canadensis
```

W linii 1 otrzymujemy skróconą informację na temat wyznaczonego wzorca odmiany: kolor niebieski oznacza, że dany segment jest odmienny – w tym przypadku mamy więc oba segmenty odmienne. Dodatkowo podkreślenie litery "w" oznacza, że dokonano korekty pisowni pierwszej litery na podstawie pisowni linków przychodzących (patrz podrozdział 8.1.2). Dalsza część linii przedstawia utworzony wzorzec syntaktyczny wyrazu. W tym przypadku znajdujemy potwierdzenie, że oba segmenty są odmienne.

Kolejne sekcje odpowiadają wystąpieniom hasła w linkach i treści artykułu, pogrupowanym według formy tekstowej. Przed każdą formą tekstową znajdziemy liczbę w kolorze niebieskim – oznacza ona ilość wystąpień danej formy w linkach. Jeżeli pewna forma wystąpiła nie tylko w linkach, po lewej stronie od niej będzie zapisana łączna liczba wystąpień – np. w linii 8 widzimy, że forma "wydry kanadyjskie" wystąpiła 2 razy, w tym 1 raz jako link. Każda forma tekstowa może zostać zapisana w kilku różnych kolorach.

– Kolor zielony oznacza, że forma pasuje do wzorca odmiany. Krojem **pogrubionym** oznaczono formy inne niż forma podstawowa, czyli pozwalające na wyznaczenie wzorca odmiany. W powyższym przykładzie formy pasujące do wzorca wypisane są w liniach 2, 4, 8, 11 i 13, a formy inne niż f. podstawowa opisują linie 4, 8 i 13. Po każdej z nich w nawiasach kwadratowych znajdziemy listę możliwych numerów form fleksyjnych, np. forma "wydry kanadyjskie" może być M, B lub W l.mn. (formy 8, 11, 14).

– Kolor pomarańczowy oznacza formy, dla których udało się wyznaczyć wzorzec odmiany, lecz został on później odrzucony przez algorytm wyznaczania optymalnego przecięcia wzorców odmiany (podrozdział 8.1.4). Przykładem może być link "wydra kanadyjskie" w linii 15.

– Kolor czerwony oznacza te linki[12], które w ogóle nie pasują do hasła, np. "Lontra canadensis" w linii 16.

Liczby zapisane kolorem jasnoniebieskim w nawiasie na końcu każdej z form oznaczają liczbę wystąpień z niepustym kontekstem, np. link "wydra kanadyjska" występuje 6 razy, ale tylko raz z niepustym kontekstem, stąd oznaczenie (1) na końcu linii 2.

Dla każdej z form pasujących do wyznaczonego wzorca odmiany wypisana jest szczegółowa statystyka wystąpień wraz z wyznaczonymi wzorcami lewego i prawego kontekstu. Przeanalizujmy ją na przykładzie linii 3. Na początku znajduje się wzorzec lewego kontekstu "g0", który wyznaczono na podstawie

[12] Do tej kategorii należą wyłącznie linki, ponieważ inne wystąpienia z definicji pasują do hasła.

segmentu "oraz"[13]. W następnej kolejności wypisywana jest forma tekstowa – kolor niebieski oznacza wystąpienie w linku. Kolejne elementy to prawy kontekst (w tym wypadku kropka) oraz wzorzec prawego kontekstu "_p". Dane wystąpienie może być też wypisane kolorem zielonym – oznacza to wystąpienie w treści bieżącego artykułu. Przykłady takich wystąpień znajdują się w liniach 12 i 14. Kolor różowy oznacza z kolei wystąpienie w korpusie WIKI[14]. Przykład takiego wystąpienia znajduje się w linii 10.

Aby utworzyć wzorce odmiany dla wszystkich haseł Wikipedii, należy użyć funkcji `patdb.create`. Podobnie jak pozostałe funkcje z modułu `patdb`, działa ona w sposób równoległy. Wynik poniższych poleceń zależy od bazy źródłowej – gdy jest nią *labels*, wynikowe wzorce uwzględniają wystąpienia haseł w linkach i bieżącym artykule, natomiast jeżeli bazą źródłową jest *pdwiki* (patrz przykład z początku niniejszego podrozdziału), wynikowe wzorce uwzględniają też wystąpienia w całej Wikipedii. Dla uproszczenia załóżmy, że źródłową bazą jest *labels*.

```
>>> db.cp('pattern'); db.set('pattern')  # wynik trafi do bazy 'pattern'
>>> patdb.create()  # tworzenie wzorców, może trwać ok. godziny

# sprawdźmy teraz wynikowe wzorce
>>> Word.objects.filter(is_entry=True).count()
176506  # liczba haseł z wzorcami

>>> Word.objects.filter(is_entry=True).exclude(ft_pattern=None).count()
173763  # liczba haseł z jednoznacznymi wzorcami

>>> w = Word.objects.get(name=u'wydra kanadyjska')

>>> w.decl, w.idfs  # wzorzec odmiany
(u'**', u'ADA-wydra:1 CC-kanadyjski:15')

>>> w.score, w.int_score  # współczynniki jakości
(0.8, 5)  #  wartości 'score' i 'int_score'
```

Porównując wypisane wartości z wcześniejszą analizą hasła "Wydra kanadyjska" zauważamy, że oba segmenty zostały słusznie oznaczone jako odmienne. Współczynnik jakości *score* wynosi 0.8, ponieważ spośród 5 linków, dla których przypadek jest inny niż mianownik, 4 są zgodne ze wzorcem. Współczynnik *int_score* wynosi 5, ponieważ oprócz tych 4 linków zliczone jest też wystąpienie "wydry kanadyjskiej" z treści artykułu (linia 14 w poprzednim przykładzie). Wystąpienie "wydry kanadyjskie" z linii 10 nie jest zliczane, ponieważ tym razem bazą źródłową jest *labels*, czyli wystąpienia w korpusie WIKI nie są brane pod uwagę.

Mając wyznaczone wzorce odmiany można przystąpić do testów metody pDM. Podobnie jak dla DM, automat można utworzyć ręcznie dla wybranych haseł Wikipedii bądź automatycznie dla wszystkich

[13] Lewy i prawy kontekst ograniczane są do określonej długości kontekstu, domyślnie długość ta wynosi 1 dla obu kontekstów.
[14] Jeżeli jakaś forma występuje zarówno w korpusie WIKI jak i w linkach lub w treści artykułu, wystąpienie w korpusie jest ignorowane celem uniknięcia duplikacji.

haseł. Sam proces ekstrakcji również można przeprowadzić dla wybranego tekstu bądź od razu dla całego korpusu. Poniższy przykład pokazuje tworzenie automatu ze wszystkich haseł i ekstrakcję na całym korpusie PAP. Bazą źródłową jest *pattern*, natomiast wynikowy automat nazwiemy *pdm*. Korpusu PAP nie trzeba tworzyć, ponieważ znajdował się on już wcześniej w bazie *pattern*[15].

```
>>> patdb.create_word_dmap('pdm')  # utwórz automat 'pdm'

>>> db.cp('pdm-test'); db.set('pdm-test')  # nowa baza danych dla testu

# operacja ekstrakcji, może potrwać kilka minut - postęp pokazują logi
>>> patdb.dmatch('pdm', corpus='pap')

# wypisz wyniki ekstrakcji dla przykładowej notatki nr 20202
>>> pmi.corpus_entry_info('#020202')
```

```
1  Marszałek Sejmu
2          12 Marszałek Sejmu
3                  12 marszałek Sejmu AAA-marszałek:1 ACA-sejm:2 1
4   Maciej Płażyński
5          47 Maciej Płażyński
6                  47 Maciej Płażyński AAA-Maciej:1 AAA-Płażyński:1 1
7   zakończył trzydniową, nieoficjalną wizytę w Rosji. (...)
```

Na powyższym przykładzie widzimy, jak metoda pDM pozwoliła na ujednoznacznienie segmentów rozpoznanych wyrazów – w porównaniu z DM udało się poprawnie określić odmienność segmentów wyrazu “Maciej Płażyński”, a także pisownię wyrazu “marszałek Sejmu”.

13.6. Ekstrakcja WW metodą SM

Aby wykonać ekstrakcję WW metodą SM (rozdział 9), należy na początku wyznaczyć wzorce syntaktyczne z kontekstami. Tak naprawdę są one już obecne w bazie danych *pattern* z powyższego przykładu, ponieważ są one zapisywane razem z wzorcami odmiany w tabeli `Word` – faza 1 zapisu wzorców syntaktycznych (opisana w podrozdziale 9.4.1) została już więc wykonana. Pozostaje więc wykonanie fazy 2, czyli zapisanie wzorców do nowych tabel w bazie. Za tę operację odpowiada funkcja `patdb.create_context_patterns`. Przyjmuje ona m.in. argument `min_combined_score`, czyli wartość parametru ws_{min}[16]. W niniejszym przykładzie wykorzystamy ponownie bazę *pattern* do pokazania metody $\text{SM}_{1,1}$. Ponieważ dla tej metody najlepszy wynik osiągnięto dla $ws_{min} = 0$, możemy nie podawać wartości `min_combined_score` – domyślnie wynosi ona właśnie 0.

[15] A konkretnie: w jednym z powyższych przykładów utworzono go w bazie *labels*, z której powstała baza *pattern*.
[16] Parametry opisano w podrozdziale 9.7.

```
>>> patdb.create_context_patterns()  # zapis wzorców, może trwać kilka minut

>>> Pattern.objects.count()  # liczba wzorców
30764
>>> ContextPattern.objects.count()  # liczba wzorców z kontekstem
560994

# przeanalizujmy wzorce utworzone z hasła 'Wydra kanadyjska'
>>> w = Word.objects.get(name=u'wydra kanadyjska')
>>> for cp in w.cps.all(): info(cp)
...
```

```
1 ac9 *ad1 *cc15_p [1:6 2,3,6:299 2:327 8,11,14:3 4:1 5:2 9:97 12:3] 738
2 p *ad1 *cc15 g [1:5849 2,3,6:575 2:379 1,7:1 8,11,14:299 4:343 10:3 5:151 9:174
3 3,6:94 12:25 13:29] 7922
4 |*ad1 *cc15 p [1:7324 2,3,6:5 2:4 8,11,14:228 4:11 5:6 9:1 3,6:1 12:1] 7581
5 p *ad1 *cc15_p [1:6227 2,3,6:1607 2:770 9:287 4:707 10:6 5:241 8,11,14:637
6 3,6:134 12:27 13:55] 10698
7 g0 *ad1 *cc15_p [1:1586 2,3,6:879 2:991 9:443 4:785 10:12 5:512 8,11,14:561
8 3,6:223 12:78 13:85] 6155
```

Wypisane wzorce kontekstowe dla wyrazu “wydra kanadyjska” pokrywają się z przedstawionym w poprzednim podrozdziale przykładem wyznaczania wzorca odmiany. W każdym wzorcu z kontekstem pogrubiony jest wzorzec wyrazu. Po wzorcu wypisana jest kolorem jasnoniebieskim statystyka wystąpień danego wzorca (atrybut *forms*, patrz podrozdział 9.4) i kolorem zielonym jego łączna liczba wystąpień (atrybut *count*). Duże liczby wynikają z faktu, że wzorce te agregują liczby wystąpień wielu różnych haseł – listę haseł źródłowych dla danego wzorca z kontekstem `cp` można łatwo pobrać z bazy danych wyrażeniem `cp.word_set.all()`.

Mając dane wzorce syntaktyczne możemy teraz przystąpić do ekstrakcji WW metodą SM. W pierwszej kolejności trzeba utworzyć automat rozpoznający wzorce. Służy do tego funkcja `patdb.create_map`. Jednym z przyjmowanych argumentów jest `min_pattern_count`, czyli wartość minimalnej liczby wystąpień wzorca wyrazu pc_{min} (patrz podrozdział 9.7). Ponieważ metoda $\mathrm{SM}_{1,1}$ osiąga najlepszy wynik dla $pc_{min} = 100$, przekażemy tę wartość tworząc automat. Podobnie jak dla metod DM i pDM istnieje możliwość ekstrakcji WW z podanego fragmentu tekstu lub z całego korpusu. Poniżej zaprezentowano pierwszą możliwość. Zwróćmy uwagę, że wykorzystywany jest moduł `pm` (*pattern matching*), a nie `dpm`, jak dla metod DM i pDM.

```
# utwórz automat 'sm-100'
>>> patdb.create_map('sm-100', min_pattern_count=100)
```

```
>>> ce = CorpusEntry(text=u'Starcie o mistrzostwo świata w kategorii
↪ półśredniej.')
>>> pmap = pm.MDBMap('sm-100')  # wczytaj automat z bazy LMDB
>>> mm = pm.MultiMatcher(pmap)  # utwórz obiekt wykonujący ekstrakcję
>>> mm.match(ce)                # wykonaj ekstrakcję
>>> info(mm.results)            # wypisz wyniki ekstrakcji
```

```
1  Starcie o mistrzostwo świata
2             10 mistrzostwo świata
3                 10 mistrzostwo świata AEA-mistrzostwo:1 ACA-świat:2 1,4,7
4                     6 g4,g6 *ae1 ac2 g 1,4,7
5                     2 g4,g6 *ae1 ac2,ac6,ac7 g 1,4,7
6                     2 g4 *ae1 ac2 g 1,4,7
7   w kategorii półśredniej
8           10353 kategorii półśredniej
9               10353 kategoria półśrednia ADA-kategoria:1 CC-półśredni:15 2,3,6
10                  10319 g4,g6 *ad1 *cc15_p 2,3,6
11                     25 g6 *ad1 *cc15_p 2,3,6
12                      3 g4,g6 *ad1 *ad1,cc15_p 2,3,6
13                      3 g4,g6 *aa1,ab1,ad1 *cc15_p 3,6
14                      2 g4,g2,g5 *ad1 *cc15_p 2,3,6
15                      1 g4,g5 *ad1 *cc15_p 3,6
```

Przypadek był na tyle prosty, że rozpoznane wyrazy są jednoznaczne. Dla każdego wyniku wyrazu wypisywana jest lista wyników wzorca (`ContextPatternResult`, patrz podrozdział 9.6). Przeanalizujmy te wyniki na przykładzie wyrazu "kategoria półśrednia" w linii 9. Łączna ocena wyniku to 10353. Poniżej widzimy statystykę pokazującą rozpoznane wzorce wraz z ocenami składowymi. Dla każdego z nich kolorem zielonym pokazywana jest ocena wyniku. Następnie wypisywany jest wzorzec z kontekstem, a po nim kolorem jasnoniebieskim lista numerów form fleksyjnych, w których rozpoznano dany wzorzec. Przykładowo wzorzec `g4,g6 *ad1 *cc15_p` ma ocenę 10319, co oznacza, że tyle wystąpień tego wzorca ma zbiór numerów form zawierający podzbiór $\{2, 3, 6\}$. Powyższą statystykę można wykorzystać do sprawdzenia, jakie wzorce przyczyniły się do ekstrakcji danego wyrazu, co jest najbardziej pomocne w przypadku błędnych wyników – pozwala dotrzeć do przyczyny błędów popełnianych przez metodę SM.

13.7. Tworzenie słownika WW metodą SM

Metoda SM może posłużyć do tworzenia słownika WW. W tym celu należy w pierwszej kolejności przygotować korpus (np. WIKI lub PAP), a następnie wykonać rozpoznawanie WW na tym korpusie. Służy do tego funkcja `patdb.match`. Do tworzenia słownika potrzebne są jedynie tabele `Word` i `WordResult` (patrz podrozdział 10.1), więc aby nie wypełniać danymi pozostałych tabel, można podać argument `options=[]`, który powoduje pominięcie opcjonalnych tabel. Poniżej podano przykład ekstrakcji WW z korpusu WIKI. W przykładzie jako źródło wykorzystano bazę danych *patterns* i automat *sm-100*, natomiast

wynikowa baza nazywa się *sm-dict*.

```
>>> db.cp('sm-dict'); db.set('sm-dict') # wynik trafi do bazy 'sm-dict'
>>> ptxt.create_wiki_corpus()

# metoda SM dla korpusu 'wiki' wykonywana równolegle
# może trwać kilka godzin ze względu na rozmiar korpusu 'wiki'
>>> patdb.match('sm-100', corpus='wiki', options=[])

>>> pm.filter_word_results().count()  # liczba wyników wyrazu
22486561
>>> pm.filter_result_words().count()  # liczba wyrazów
5140923
```

Widzimy, że rozpoznano ponad 5 milionów różnych wyrazów. Należy teraz ujednoznacznić te wyrazy tak, aby dla każdej formy podstawowej wyznaczony został dokładnie jeden wyraz (patrz przykład w podrozdziale 10.1). W tym celu używamy funkcji `patdb.finalize_results`.

```
>>> patdb.finalize_results()  # ujednoznaczniane słowa są wypisywane w logach

>>> pm.filter_final_words().count()  # liczba wyrazów nieznacznie spadła
4923754

# wypisz 3 wyrazy nie będące hasłami Wikipedii z największą liczbą wystąpień
# dla każdego z nich wypisz co najwyżej 5 najczęstszych form
>>> pmi.final_word_stat_info(min_word_score=2000000, nowiki=True, limit=3,
↪  text_limit=5)
```

```
147804568  2089 styl wolen ACA-styl:1 CC-wolen:1,8|CC-wolny:1,8
        147763219  1995 stylu wolnym
            37821    14 styl wolny
             1950    75 stylem wolnym
             1487     4 Styl wolny
               91     1 stylu wolny
 56278822 10459 klasyfikacja generalna ADA-klasyfikacja:1 CC-generalny:15
         56213390 10246 klasyfikacji generalnej
            42759    36 Klasyfikacja generalna
             7864    62 klasyfikację generalną
             6931    29 Klasyfikacje generalne
             3709    13 klasyfikacja generalna
 46529161  1567 wypadek samochodowy ACA-wypadek:1 CC-samochodowy:1,8
         46008972  1127 wypadku samochodowym
           293841   172 wypadek samochodowy
           207179   140 wypadku samochodowego
             7864    17 wypadków samochodowych
             4278     4 wypadkach samochodowych
```

Powyższa statystyka pokazuje 3 najczęściej występujące WW, które nie są hasłami Wikipedii wraz z ich wystąpieniami. Kolorem różowym wypisywane są łączne oceny wyrazów i wystąpień, a jasnoniebieskim – odpowiadające im liczby wystąpień. Dla każdego wyrazu znajdziemy też informacje o odmienności segmentów oraz znaczniki słownikowe. Uwagę zwraca pierwszy wyraz (linia 1), który co prawda rozpoznano poprawnie, ale forma podstawowa "styl wolen" jest błędna. Jest to spowodowane faktem, że formy "wolen" i "wolny" są zapisane jako alternatywne w słowniku CLPM. Wyraz jest więc rozpoznany poprawnie, jedynie jego forma hasłowa jest błędna. Można to poprawić, opierając formę podstawową o najczęściej występującą formę mianownika w tekście (na podstawie linii 3 i 5).

13.8. Ekstrakcja WW metodą SDM

Słownik utworzony metodą SM można wykorzystać do ekstrakcji WW metodą SDM (podrozdział 10.2). W tym celu należy dla bazy danych zawierającej dany słownik skonstruować automat rozpoznający wzorce słownikowe. Podobnie jak dla metody pDM, używamy w tym celu funkcji `patdb.create_word_dmap`. Podczas tworzenia można podać argument `min_word_score`, który odpowiada wartości parametru rs_{min} (patrz podrozdział 10.2). Rozważmy przykład, w którym dla bazy danych *sm-dict* z poprzedniego podrozdziału tworzony jest automat rozpoznający wzorce metodą SDM$_W$. Wartość progu rs_{min} wynosi 10. Po utworzeniu automatu ekstrakcję WW wykonujemy identycznie jak dla zwykłej metody pDM – nie są konieczne żadne modyfikacje.

```
# tworzymy automat dla samej metody SDM
>>> patdb.create_word_dmap('sdm', min_word_score=10, with_entries=False)

# tworzymy automat dla metody pDM + SDM
>>> patdb.create_word_dmap('pdm+sdm', min_word_score=10)

# ekstrakcja WW metodą SDM przebiega identycznie jak dla pDM
>>> db.cp('sdm-test'); db.set('sdm-test')
>>> patdb.dmatch('sdm', corpus='pap')
```

Widzimy, że domyślnie do automatu dołączane są wzorce słownikowe tworzone z haseł Wikipedii – otrzymujemy wówczas metodę pDM + SDM$_W$. Możemy również dodać do automatu dodatkowe słowniki tworzone przy pomocy SM, tworząc np. metodę pDM + SDM$_W$ + SDM$_P$. Główny problem przy takim łączeniu metod polega na tym, że możemy scalić jedynie metody rozpoznające wzorce słownikowe. Nie uzyskamy więc metody łączonej pDM + SDM$_W$ + SDM$_P$ + SM (podrozdział 10.3), ponieważ metoda SM wykorzystuje wzorce syntaktyczne. Efekt ten możemy jednak osiągnąć wykonując oddzielnie ekstrakcję każdą z metod składowych, a następnie łącząc wyniki. Procedurę taką zastosowano podczas testów metody łączonej.

13.9. Testowanie metod ekstrakcji WW

Dotychczasowe przykłady pokazywały, jak dokonać ekstrakcji WW poszczególnymi metodami. Rezultaty ekstrakcji za każdym razem umieszczane były w bazie danych. Wyniki te mogą następnie posłużyć do zbadania skuteczności ekstrakcji na korpusie testowym PAP-WW. Sam korpus umieszczony jest w pliku `data/corp/pap-ww.utf8`. Istnieją gotowe funkcje pozwalające na przetestowanie dowolnej metody. W najprostszym przypadku test sprowadza się do przeprowadzenia ekstrakcji na korpusie testowym przy pomocy funkcji `patdb.prepare_test_match` (dla wzorców syntaktycznych) lub `patdb.prepare_test_dmatch` (dla wzorców słownikowych). Następnie należy użyć jednej z kilku funkcji z modułu `pmi` do sprawdzenia wyników ekstrakcji. Poniżej pokazano przykład, w którym wykorzystano automat *sdm* utworzony w poprzednim przykładzie.

```
# ekstrahuj wzorce z korpusu testowego 'pap-ww' metodą SDM
>>> patdb.prepare_test_dmatch(map_name='sdm')

# sprawdź wyniki testu
>>> pmi.test_match().print_row()
#  P_rec   R_rec   F_rec   P_ext   R_ext   F_ext
  52.42   71.69   60.56   50.93   69.66   58.84

# test dla różnych wartości progu rs_min
>>> pmi.test_match_ms([10, 100, 300, 500, 1000, 10000])
   10  52.42  71.69  60.56  50.93  69.66  58.84
  100  58.32  67.12  62.41  56.55  65.08  60.52
  300  62.19  64.41  63.28  60.23  62.37  61.28
  500  63.68  61.53  62.59  61.58  59.49  60.52
 1000  64.52  56.10  60.02  63.35  55.08  58.93
10000  74.15  40.85  52.68  73.23  40.34  52.02

# wyczyść wyniki ekstrakcji
>>> patdb.clean_test_match()
```

Jako drugi przykład rozważmy metodę łączoną pDM + SDM$_W$ + SDM$_P$ + SM. Załóżmy, że istnieją już automaty *pdm*, *sdm-w*, *sdm-p* i *sm* realizujące poszczególne metody składowe. Należy teraz dokonać rozpoznawania WW kolejnymi automatami. Spowoduje to zapisanie wyników do bazy z kolejnymi numerami iteracji (wartościami atrybutu *iteration*). Przy pomocy funkcji `patdb.test_match` możemy teraz sprawdzić wyniki łącząc metody. W tym celu jako argument przekazujemy słownik, w którym kluczami są numery iteracji, a wartościami – progi rs_{min} odpowiadające tym iteracjom. W poniższym przykładzie zastosowano wartości optymalne (wyznaczono je w podrozdziale 10.3).

```
# obecny numer iteracji to 1
>>> pm.get_iteration(Word)
1

# ekstrahujemy WW kolejnymi automatami
# wyniki odpowiadają iteracjom 2, 3, 4 i 5
>>> patdb.prepare_test_dmatch(map_name='pdm')
>>> patdb.prepare_test_dmatch(map_name='sdm-w')
>>> patdb.prepare_test_dmatch(map_name='sdm-p')
>>> patdb.prepare_test_match(map_name='sm')

# wartości rs_min dla kolejnych iteracji
>>> config = {2:1, 3:100000, 4:1200, 5:65}

>>> pmi.test_match(config).print_row()
  72.27  70.14  71.19  69.23  67.19  68.19
```

Istnieje możliwość obejrzenia szczegółowych rezultatów dla wybranej notatki. Przykład zaprezentowano poniżej.

```
# wypisz szczegółowe wyniki dla notatki nr 40 (numery są od 0 do 99)
>>> pmi.ce_test(40, iterations=config)
-------------------------------------------------------------------------------
#020947 Francja wyklucza wyznaczanie daty rozszerzenia Unii Europejskiej
          ** Unii Europejskiej
 o nowych członków
          ** nowych członków
, uznając że podobna decyzja byłaby "demagogiczna" - oznajmił szef francuskiej
dyplomacji Hubert Vedrine
          *- Hubert Vedrine
. Minister spraw zagranicznych
        *-- Minister spraw zagranicznych
 Francji (kraju, który w tym półroczu przewodniczy Unii Europejskiej
          ** Unii Europejskiej
) powiedział, że nie może być opóźnienia w rozszerzeniu UE, gdyż nigdy nie
ustalono żadnej daty. Polska domaga się jak najwcześniejszego wyznaczenia daty
rozszerzenia Unii Europejskiej
          ** Unii Europejskiej
, życząc sobie, żeby przyjęcie nowych członków
          ** nowych członków
 nastąpiło już 1 stycznia 2003
          *- stycznia 2003
 roku. Ostatnio premier W.Brytanii
       *--- premier W.Brytanii
 Tony Blair
```

```
          ** Tony Blair (-*)
w czasie wizyty
          *- czasie wizyty
w Warszawie wspominał o roku 2004.
          *- roku 2004
```

W pierwszej kolejności kolorem różowym wypisany jest identyfikator notatki (atrybut *key* z tabeli `CorpusEntry`). Następnie pokazywane są kolejno elementy należące do poszczególnych zbiorów tworzonych podczas testu (por. z podrozdziałami 6.2.2 i 10.3.1), oznaczane poniższymi kolorami:

– Na zielono oznaczane są elementy T_i, czyli wyrażenia rozpoznane poprawnie.

– Na pomarańczowo oznaczane są elementy T_d, czyli wyrażenia zlokalizowane poprawnie, lecz o błędnie rozpoznanej odmienności segmentów. W powyższym przykładzie widzimy, że przypadek taki nastąpił dla hasła "Tony Blair". Na czerwono w nawiasie pokazany jest niepoprawnie rozpoznany znacznik odmienności segmentów `-*`. Widzimy, że segment "Tony" został omyłkowo uznany za nieodmienny.

– Na czerwono oznaczone są elementy F_n, czyli wyrażenia brakujące, których nie rozpoznano. Przykładem jest tu wyrażenie "premier W.Brytanii", które nie zostało rozpoznane przez nietypową pisownię – brak odstępu po skrócie "W.".

– Na niebiesko oznaczono elementy F_p czyli wyrażenia błędnie rozpoznane jako WW.

Funkcjonalność wspierająca testy ekstrakcji jest dużo szersza, np. klasa `pmi.StatsDiff` pozwala na porównywanie wyników testu różnych metod, a funkcja `pmi.test_multiple` umożliwia optymalizację parametrów. Funkcje te nie będą już omawiane.

13.10. Podsumowanie

W niniejszym dodatku przedstawiono główną funkcjonalność utworzonego na potrzeby niniejszej pracy projektu. Zaprezentowano narzędzia umożliwiające ekstrakcję danych z Wikipedii, tworzenie etykiet semantycznych oraz przede wszystkim ekstrakcję WW przy użyciu różnych metod i ewaluację wyników ekstrakcji. W rzeczywistości funkcjonalność projektu jest dużo szersza – opis ograniczono do głównych funkcji. W przyszłości planowany jest dalszy rozwój projektu, prowadzący do utworzenia słowników i innych zasobów zawierających wyrazy wielosegmentowe dla języka polskiego.

Bibliografia

[1] Anne Abeillé, Lionel Clément i François Toussenel. *Building a Treebank for French*. Treebanks, tom 20 z serii *Text, Speech and Language Technology*, str. 165–187. Springer Netherlands, 2003.

[2] Szymon Acedański. *A morphosyntactic Brill tagger for inflectional languages*. Advances in Natural Language Processing, tom 6233 z serii *Lecture Notes in Computer Science*, str. 3–14. Springer Berlin Heidelberg, 2010.

[3] Mohammed Attia, Lamia Tounsi, Pavel Pecina, Josef van Genabith i Antonio Toral. *Automatic extraction of Arabic multiword expressions*. 23rd International Conference on Computational Linguistics: Proceedings of the Workshop on Multiword Expressions: From Theory to Applications (MWE), str. 19–27. Association for Computational Linguistic, 2010.

[4] P. V. S. Avinesh i G. Karthik. *Part-of-speech tagging and chunking using conditional random fields and transformation based learning*. Proceedings of the IJCAI and the Workshop On Shallow Parsing for South Asian Languages (SPSAL), str. 21–24. 2007.

[5] Božo Bekavac i Marko Tadic. *A generic method for multi word extraction from Wikipedia*. 30th International Conference on Information Technology Interfaces (ITI), str. 663–668. IEEE, 2008.

[6] Thorsten Brants. *TnT: a statistical part-of-speech tagger*. Proceedings of the 6th Conference on Applied Natural Language Processing, str. 224–231. Association for Computational Linguistics, 2000.

[7] Eric Brill. *Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging*. Computational linguistics, 21(4), str. 543–565, 1995.

[8] Aleksander Buczyński i Adam Przepiórkowski. *Spejd: A shallow processing and morphological disambiguation tool*. Human Language Technology. Challenges of the Information Society, str. 131–141. Springer, 2009.

[9] Noam Chomsky. *Aspects of the theory of syntax*. MIT Press, 1965.

[10] Paweł Chrząszcz. *Automatyczne rozpoznawanie i klasyfikacja nazw wielosegmentowych na podstawie analizy haseł encyklopedycznych*. Praca magisterska, Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie, 2009.

[11] Paweł Chrząszcz. *Enrichment of inflection dictionaries: automatic extraction of semantic labels from encyclopedic definitions*. Proceedings of the 9th International Workshop on Natural Language Processing and Cognitive Science (NLPCS, w połączeniu z ICEIS), str. 106–119. SciTePress, 2012.

[12] Paweł Chrząszcz. *Extraction of Polish multiword expressions*. Proceedings of the 11th International Workshop on Natural Language Processing and Cognitive Science (NLPCS), str. 245–256. De Gruyter, 2014.

[13] Kenneth Church. *How many multiword expressions do people know?* ACM Transactions on Speech and Language Processing (TSLP), 10(2), str. 4, 2013.

[14] Matthieu Constant i Anthony Sigogne. *MWU-aware part-of-speech tagging with a CRF model and lexical resources*. Proceedings of the Workshop on Multiword Expressions: from Parsing and Generation to the Real World, str. 49–56. Association for Computational Linguistics, 2011.

[15] Matthieu Constant, Anthony Sigogne i Patrick Watrin. *Discriminative strategies to integrate multiword expression recognition and parsing*. Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers – Volume 1, str. 204–212. Association for Computational Linguistics, 2012.

[16] Blandine Courtois, Mylène Garrigues, Gaston Gross, Maurice Gross, René Jung, Michel Mathieu-Colas, Anne Monceaux, Anne Poncet-Montange, Max Silberztein i Robert Vivés. *Dictionnaire électronique DELAC: les mots composés binaires*. Rapport technique, 56, 1997.

[17] *Django. The Web framework for perfectionists with deadlines*. Dostępny 8 maja 2015. `https://www.djangoproject.com`

[18] Philip Edmonds i Adam Kilgarriff. *Introduction to the special issue on evaluating word sense disambiguation systems*. Natural Language Engineering, 8(4), str. 279–291, 2002.

[19] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S Weld i Alexander Yates. *Unsupervised named-entity extraction from the web: An experimental study*. Artificial intelligence, 165(1), str. 91–134, 2005.

[20] Meghdad Farahmand i Ronaldo Martins. *A supervised model for extraction of multiword expressions based on statistical context features*. Proceedings of the 10th Workshop on Multiword Expressions (MWE, w połączeniu z EACL), str. 10–16. Association for Computational Linguistics, 2014.

[21] Christiane Fellbaum. *WordNet: an electronic lexical database*. MIT Press, 1998.

[22] Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman i Eytan Ruppin. *Placing search in context: The concept revisited*. Proceedings of the 10th international conference on World Wide Web, str. 406–414. ACM, 2001.

[23] Radu Florian, Abe Ittycheriah, Hongyan Jing i Tong Zhang. *Named entity recognition through classifier combination*. Proceedings of the 7th Conference on Natural Language Learning at HLT-NAACL 2003 – Volume 4, str. 168–171. Association for Computational Linguistics, 2003.

[24] Radu Florian i Grace Ngai. *fnTBL, fast transformation-based learning toolkit*, 2001. Dostępny 4 czerwca 2015.
`https://www.cs.jhu.edu/~rflorian/fntbl`

[25] Evgeniy Gabrilovich i Shaul Markovitch. *Computing semantic relatedness using Wikipedia-based explicit semantic analysis.* Proceedings of the 20th International Joint Conference on Artifical Intelligence (IJCAI), tom 7, str. 1606–1611. Morgan Kaufmann Publishers Inc., 2007.

[26] Marek Gajęcki. *Słownik fleksyjny jako biblioteka języka C*. Słowniki komputerowe i automatyczna ekstrakcja informacji z tekstu (pod redakcją Wiesława Lubaszewskiego). Wydawnictwa AGH, Kraków, 2009.

[27] Zoubin Ghahramani. *An introduction to hidden Markov models and Bayesian networks*. International Journal of Pattern Recognition and Artificial Intelligence, World Scientific, 15(01), str. 9–42, 2001.

[28] Jesús Giménez i Lluis Marquez. *SVMTool: A general POS tagger generator based on Support Vector Machines*. Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC), str. 43–46. European Language Resources Association, 2004.

[29] Filip Graliński, Agata Savary, Monika Czerepowicka i Filip Makowiecki. *Computational lexicography of multi-word units: how efficient can it be?* Proceedings of the Workshop on Multiword Expressions: from Theory to Applications (MWE), str. 1–9. Association for Computational Linguistics, 2010.

[30] Marti A. Hearst, Susan T. Dumais, Edgar Osman, John Platt i Bernhard Scholkopf. *Support vector machines*. Intelligent Systems and their Applications, IEEE, 13(4), str. 18–28, 1998.

[31] *Jenkins CI. An extensible open source continuous integration server.* Dostępny 8 maja 2015.
`https://jenkins-ci.org`

[32] Michael I. Jordan i Andrew Y. Ng. *On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes*. Advances in neural information processing systems, MIT, 14, str. 841–848, 2002.

[33] Jun'ichi Kazama i Kentaro Torisawa. *Exploiting Wikipedia as external knowledge for named entity recognition.* Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), str. 698–707. ACL, 2007.

[34] J-D Kim, Tomoko Ohta, Yuka Tateisi i Jun'ichi Tsujii. *GENIA corpus—a semantically annotated corpus for bio-textmining*. Bioinformatics, Oxford Univ Press, 19(suppl 1), str. i180–i182, 2003.

[35] Łukasz Kobyliński. *PoliTa: a Multitagger for Polish*. Proceedings of the 9th International Conference on Language Resources and Evaluation (LREC), str. 2949–2954. European Language Resources Association, 2014.

[36] Philipp Koehn. *Europarl: A parallel corpus for statistical machine translation*. Machine Translation Summit X, tom 5, str. 79–86. 2005.

[37] Taku Kudo. *CRF++: Yet another CRF toolkit*, 2005. Dostępny 6 czerwca 2015. `http://sourceforge.net/projects/crfpp`

[38] Roman Kurc, Maciej Piasecki i Bartosz Broda. *Constraint based description of Polish multiword expressions*. Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC), str. 2408–2413. European Language Resources Association, 2012.

[39] Marcin Kuta, Paweł Chrząszcz i Jacek Kitowski. *A case study of algorithms for morphosyntactic tagging of Polish language*. Computing and Informatics, 26(6), str. 627–647, 2007.

[40] Marcin Kuta, Jacek Kitowski, Wojciech Wójcik i Michał Wrzeszcz. *Application of weighted voting taggers to languages described with large tagsets*. Computing and Informatics, 29(2), str. 203–225, 2010.

[41] Thomas K. Landauer, Peter W. Foltz i Darrell Laham. *An introduction to latent semantic analysis*. Discourse processes, Taylor & Francis, 25(2-3), str. 259–284, 1998.

[42] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer i in. *DBpedia – a large-scale, multilingual knowledge base extracted from Wikipedia*. Semantic Web, IOS Press, 6(2), str. 167–195, 2015.

[43] *LMDB. Symas Lightning Memory-Mapped Database*. Dostępny 8 maja 2015. `http://symas.com/mdb`

[44] Wiesław Lubaszewski. *Wyraz*. Słowniki komputerowe i automatyczna ekstrakcja informacji z tekstu (pod redakcją Wiesława Lubaszewskiego). Wydawnictwa AGH, Kraków, 2009.

[45] Wiesław Lubaszewski, H. Wróbel, M. Gajęcki, B. Moskal, A. Orzechowska, P. Pietras, P. Pisarek i T. Rokicka. *Słownik Fleksyjny Języka Polskiego*. Grupa Lingwistyki Komputerowej, Katedra Informatyki AGH i Katedra Lingwistyki Komputerowej UJ, Kraków, 2001.

[46] Michał Marcińczuk. *NELexicon. Gazetteer nazw własnych*, 2012. Dostępny 6 czerwca 2015. `http://hdl.handle.net/11321/22`

[47] Małgorzata Marciniak, Agata Savary, Piotr Sikora i Marcin Woliński. *Toposław – a lexicographic framework for multi-word units*. Human Language Technology. Challenges for Computer Science and Linguistics, tom 6562 z serii *Lecture Notes in Computer Science*, str. 139–150. Springer Berlin Heidelberg, 2011.

[48] Cynthia Matuszek, John Cabral, Michael J Witbrock i John DeOliveira. *An introduction to the syntax and content of Cyc*. AAAI Spring Symposium: Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering, str. 44–49. 2006.

[49] Marek Maziarz, Maciej Piasecki i Stanisław Szpakowicz. *Approaching plWordNet 2.0*. Proceedings of the 6th Global Wordnet Conference. Global WordNet Association, 2012.

[50] Andrew McCallum i Wei Li. *Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons*. Proceedings of the 7th Conference on Natural Language Learning at HLT-NAACL 2003 – Volume 4, str. 188–191. Association for Computational Linguistics, 2003.

[51] Olena Medelyan, David Milne, Catherine Legg i Ian H. Witten. *Mining meaning from Wikipedia*. International Journal of Human-Computer Studies, Academic Press Inc., 67(9), str. 716–754, 2009.

[52] *MediaWiki. A free software open source wiki package written in PHP*. Dostępny 23 maja 2015. `https://www.mediawiki.org/wiki/MediaWiki`

[53] Rada Mihalcea i Andras Csomai. *Wikify!: linking documents to encyclopedic knowledge*. Proceedings of the 16th Conference on Information and Knowledge Management (CIKM), str. 233–242. Association for Computing Machinery, 2007.

[54] Rada Mihalcea i Dan I Moldovan. *EZ. WordNet: principles for automatic generation of a coarse grained WordNet*. Proceedings of the 14th International Florida Artificial Intelligence Research Society Conference (FLAIRS), str. 454–458. AAAI Press, 2001.

[55] David Milne i Ian H. Witten. *Learning to link with wikipedia*. Proceedings of the 17th Conference on Information and Knowledge Management (CIKM), str. 509–518. Association for Computing Machinery, 2008.

[56] Marcin Miłkowski i Dawid Weiss. *Morfologik stemming library including precompiled Polish dictionaries*. Dostępny 8 maja 2015. `https://github.com/morfologik/morfologik-stemming`

[57] *Morfologik. Analizator morfologiczny + słownik morfologiczny + korektor gramatyczny + biblioteki*. Dostępny 8 maja 2015. `http://morfologik.blogspot.com`

[58] Radosław Moszczyński. *A practical classification of multiword expressions*. Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop, str. 19–24. Association for Computational Linguistics, 2007.

[59] *Mwlib. Mediawiki parser and utility library*. Dostępny 23 maja 2015. `https://pypi.python.org/pypi/mwlib`

[60] David Nadeau i Satoshi Sekine. *A survey of named entity recognition and classification*. Lingvisticae Investigationes, John Benjamins publishing company, 30(1), str. 3–26, 2007.

[61] David Nadeau, Peter D. Turney i Stan Matwin. *Unsupervised named-entity recognition: generating gazetteers and resolving ambiguity*. Advances in Artificial Intelligence, tom 4013 z serii *Lecture Notes in Computer Science*, str. 266–277. Springer Berlin Heidelberg, 2006.

[62] Joel Nothman, Tara Murphy i James R Curran. *Analysing Wikipedia and gold-standard corpora for NER training*. Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics, str. 612–620. Association for Computational Linguistics, 2009.

[63] Petr Pajas i Jan Štěpánek. *Recent advances in a feature-rich framework for treebank annotation*. Proceedings of the 22nd International Conference on Computational Linguistics – Volume 1, str. 673–680. Association for Computational Linguistics, 2008.

[64] Pavel Pecina. *A machine learning approach to multiword expression extraction*. Proceedings of the LREC Workshop – Towards a Shared Task for Multiword Expressions (MWE), str. 54–61. European Language Resources Association, 2008.

[65] Jakub Piskorski. *Named-entity recognition for Polish with SProUT*. Intelligent Media Technology for Communicative Intelligence, tom 3490 z serii *Lecture Notes in Computer Science*, str. 122–133. Springer Berlin Heidelberg, 2005.

[66] Jakub Piskorski, Peter Homola, Małgorzata Marciniak, Agnieszka Mykowiecka, Adam Przepiórkowski i Marcin Woliński. *Information extraction for Polish using the SProUT platform*. Intelligent Information Processing and Web Mining, tom 25 z serii *Advances in Soft Computing*, str. 227–236. Springer Berlin Heidelberg, 2004.

[67] Aleksander Pohl. *Słownik Semantyczny Języka Polskiego*. Słowniki komputerowe i automatyczna ekstrakcja informacji z tekstu (pod redakcją Wiesława Lubaszewskiego). Wydawnictwa AGH, Kraków, 2009.

[68] Aleksander Pohl. *Knowledge-based named entity recognition in Polish*. Federated Conference on Computer Science and Information Systems (FedCSIS 2013), str. 145–151. IEEE, 2013.

[69] Aleksander Pohl i Bartosz Ziółko. *A comparison of Polish taggers in the application for automatic speech recognition*. Proceedings of the 6th Language and Technology Conference (LTC), str. 294–298. 2013.

[70] Simone Paolo Ponzetto i Michael Strube. *Knowledge derived from wikipedia for computing semantic relatedness.* Journal of Artificial Intelligence Research (JAIR), 30, str. 181–212, 2007.

[71] Adam Przepiórkowski, Mirosław Bańko, Rafał Górski i Barbara Lewandowska-Tomaszczyk. *Narodowy Korpus Języka Polskiego*. Wydawnictwo Naukowe PWN, 2012.

[72] Adam Przepiórkowski. *Korpus IPI PAN. Wersja wstępna*. Instytut Podstaw Informatyki, Polska Akademia Nauk, Warszawa, 2004.

[73] *Python. A programming language that lets you work quickly and integrate systems more effectively*. Dostępny 8 maja 2015.
`https://www.python.org`

[74] Adam Radziszewski. *A tiered CRF tagger for Polish*. Intelligent Tools for Building a Scientific Information Platform (pod redakcją Robert Bembenik, Lukasz Skonieczny, Henryk Rybinski, Marzena Kryszkiewicz i Marek Niezgodka), tom 467 z serii *Studies in Computational Intelligence*, str. 215–230. Springer Berlin Heidelberg, 2013.

[75] Adam Radziszewski i Tomasz Śniatowski. *Maca – a configurable tool to integrate Polish morphological data*. Proceedings of the 2nd International Workshop on Free/Open-Source Rule-Based Machine Translation, str. 29–36. 2011.

[76] Adam Radziszewski i Tomasz Śniatowski. *A memory-based tagger for Polish*. 5th Language and Technology Conference (LTC). 2011.

[77] Carlos Ramisch, Paulo Schreiner, Marco Idiart i Aline Villavicencio. *An evaluation of methods for the extraction of multiword expressions*. Proceedings of the LREC Workshop – Towards a Shared Task for Multiword Expressions (MWE), str. 50–53. European Language Resources Association, 2008.

[78] Carlos Ramisch, Aline Villavicencio i Christian Boitet. *MWEToolkit: a framework for multiword expression identification*. Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC), str. 662–669. European Language Resources Association, 2010.

[79] Juan Ramos. *Using TF-IDF to determine word relevance in document queries*. Raport techniczny, Department of Computer Science, Rutgers University, Piscataway, NJ, 2003.

[80] Lev Ratinov i Dan Roth. *Design challenges and misconceptions in named entity recognition*. Proceedings of the 13th Conference on Computational Natural Language Learning, str. 147–155. Association for Computational Linguistics, 2009.

[81] Adwait Ratnaparkhi i in. *A maximum entropy model for part-of-speech tagging*. Proceedings of the Conference on Empirical Methods in Natural Language Processing, tom 1, str. 133–142. 1996.

[82] *repoze.lru – a tiny LRU cache implementation and decorator*. Dostępny 8 maja 2015. `https://pypi.python.org/pypi/repoze.lru`

[83] Alexander E. Richman i Patrick Schone. *Mining wiki resources for multilingual named entity recognition*. Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, str. 1–9. 2008.

[84] Emmanuel Roche i Yves Schabes. *Finite-state language processing*. MIT Press, 1997.

[85] Josef Ruppenhofer, Michael Ellsworth, Miriam R.L. Petruck, Christopher R. Johnson i Jan Scheffczyk. *FrameNet II: Extended theory and practice*. International Computer Science Institute, Berkeley, CA, 2006.

[86] Ivan A. Sag, Timothy Baldwin, Francis Bond, Ann Copestake i Dan Flickinger. *Multiword expressions: a pain in the neck for NLP*. Computational Linguistics and Intelligent Text Processing, tom 2276 z serii *Lecture Notes in Computer Science*, str. 1–15. Springer Berlin Heidelberg, 2002.

[87] Benoît Sagot. *The Lefff, a freely available, accurate and large-coverage lexicon for French*. Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC), str. 2744–2751. European Language Resources Association, 2010.

[88] Agata Savary. *Multiflex: a multilingual finite-state tool for multi-word units*. Implementation and Application of Automata, tom 5642 z serii *Lecture Notes in Computer Science*, str. 237–240. Springer Berlin Heidelberg, 2009.

[89] Agata Savary i Jakub Piskorski. *Language resources for named entity annotation in the National Corpus of Polish*. Control and Cybernetics, 40, str. 361–391, 2011.

[90] Agata Savary, Jakub Waszczuk i Adam Przepiórkowski. *Towards the Annotation of Named Entities in the National Corpus of Polish.* Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC), str. 3622–3629. European Language Resources Association, 2010.

[91] Agata Savary i Bartosz Zaborowski. *SEJFEK – a lexicon and a shallow grammar of Polish economic multi-word units*. Proceedings of the 3rd Workshop on Cognitive Aspects of the Lexicon (CogALex-III), 24th International Conference on Computational Linguistic (COLING), str. 195–214. 2012.

[92] Roger C. Schank. *Conceptual information processing*. Elsevier Science Inc., 1975.

[93] *SQLite. A self-contained, serverless, zero-configuration, transactional SQL database engine*. Dostępny 8 maja 2015. `https://www.sqlite.org`

[94] Fabian M. Suchanek, Gjergji Kasneci i Gerhard Weikum. *YAGO: a large ontology from Wikipedia and WordNet*. Journal of Web Semantics, 6(3), str. 203–217, 2008.

[95] Charles Sutton i Andrew McCallum. *An introduction to conditional random fields*. Foundations and Trends in Machine Learning, 4(4), str. 267–373, 2012.

[96] *syslog-ng – open source log management solution with over a million global users, an enhanced syslog daemon: the Babel fish of event processing*. Dostępny 8 maja 2015. `https://syslog-ng.org`

[97] Erik F. Tjong Kim Sang i Fien De Meulder. *Introduction to the CoNLL-2003 shared task: language-independent named entity recognition*. Proceedings of the 7th Conference on Natural Language Learning at HLT-NAACL 2003 – Volume 4, str. 142–147. Association for Computational Linguistics, 2003.

[98] Maksim Tkachenko, Alexander Ulanov i Andrey Simanovsky. *Fine grained classification of named entities in Wikipedia*. HP Laboratories Technical Report – HPL-2010-166, 2010.

[99] A. Toral i R. Muñoz. *A proposal to automatically build and maintain gazetteers for Named Entity Recognition by using Wikipedia*. Proceedings of the workshop on NEW TEXT – Wikis and blogs and other dynamic text sources, str. 56–61. Association for Computational Linguistics, 2006.

[100] Peter D. Turney. *Mining the Web for Synonyms: PMI-IR versus LSA on TOEFL*. Machine Learning: ECML 2001, tom 2167 z serii *Lecture Notes in Computer Science*, str. 491–502. Springer Berlin Heidelberg, 2001.

[101] Veronika Vincze, István Nagy i Gábor Berend. *Multiword expressions and named entities in Wikipedia articles*. Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP), str. 289–295. 2011.

[102] Ellen M. Voorhees. *Natural language processing and information retrieval*. Information Extraction, tom 1714 z serii *Lecture Notes in Computer Science*, str. 32–48. Springer Berlin Heidelberg, 1999.

[103] Jakub Waszczuk. *Harnessing the CRF complexity with domain-specific constraints. The case of morphosyntactic tagging of a highly inflected language.* Proceedings of the 24th International Conference on Computational Linguistics (COLING), str. 2789–2804. 2012.

[104] Jakub Waszczuk, Katarzyna Głowińska, Agata Savary i Adam Przepiórkowski. *Tools and methodologies for annotating syntax and named entities in the National Corpus of Polish*. Proceedings of the 2010 International Multiconference on Computer Science and Information Technology (IMCSIT), str. 531–539. IEEE, 2010.

[105] *Wikipedia. Wolna encyklopedia*. Dostępny 8 maja 2015. `https://pl.wikipedia.org`

[106] *Wikisłownik. Wolny, wielojęzyczny słownik*. Dostępny 23 maja 2015. `http://pl.wiktionary.org`

[107] Marcin Woliński. *Morfeusz — a practical tool for the morphological analysis of Polish*. Advances in Soft Computing, 26(6), str. 503–512, 2006.

[108] Marcin Woliński, Marcin Miłkowski, Maciej Ogrodniczuk i Adam Przepiórkowski. *PoliMorf: a (not so) new open morphological dictionary for Polish.* Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC), str. 860–864. European Language Resources Association, 2012.

[109] Michał Woźniak. *Automatic extraction of multiword lexical units from Polish text*. 5th Language and Technology Conference (LTC). 2011.

[110] Tong Zhang i David Johnson. *A robust risk minimization based named entity recognition system*. Proceedings of the 7th Conference on Natural Language Learning at HLT-NAACL 2003 – Volume 4, str. 204–207. Association for Computational Linguistics, 2003.

[111] Yi Zhang, Valia Kordoni, Aline Villavicencio i Marco Idiart. *Automated multiword expression prediction for grammar engineering*. Proceedings of the Workshop on Multiword Expressions: Identifying and Exploiting Underlying Properties, str. 36–44. Association for Computational Linguistics, 2006.

[112] GuoDong Zhou i Jian Su. *Named entity recognition using an HMM-based chunk tagger*. Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, str. 473–480. Association for Computational Linguistics, 2002.

[113] Leonardo Zilio, Luiz Svoboda, Luiz Henrique Longhi Rossi i Rafael Martins Feitosa. *Automatic extraction and evaluation of MWE*. 8th Brazilian Symposium in Information and Human Language Technology, str. 214–218. 2011.