

Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Aleksander Buczyński

Nr albumu: 155542

Pozyskiwanie z Internetu tekstów do badań lingwistycznych

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
dra hab. Janusza S. Bienia
Katedra Lingwistyki Formalnej UW

Luty 2004

Pracę przedkładam do oceny

Data

Podpis autora pracy:

Praca jest gotowa do oceny przez recenzenta

Data

Podpis kierującego pracą:

Streszczenie

W pracy przedstawiono opis realizacji programu Kolokacje, służącego do pozyskiwania z Internetu tekstów do potrzeb badań lingwistycznych, a także wykrywania kolokacji w pozyskanych zbiorach tekstów za pomocą zestawu testów statystycznych oraz do porównywania działania tych testów.

Program został napisany w języku Java, opcjonalny interfejs WWW wykorzystuje dodatkowo technologię PHP. Program Kolokacje dostępny jest bezpłatnie na licencji GNU General Public Licence i składa się z wyraźnie wyodrębnionych modułów, co daje duże możliwości dalszego rozwoju lub wykorzystywania jego elementów w innych programach.

Tekst pracy dostępny jest na licencji GNU Free Documentation License.

Słowa kluczowe

Internet, roboty internetowe, korpusy, słowniki, kolokacje, ekscerpca

Klasyfikacja tematyczna

H. Information Systems
H.3 Information Storage and Retrieval
H.3.1 Content Analysis and Indexing
Linguistic processing
Także:
J. Computer Applications
J.5 Arts and humanities
Linguistics

Spis treści

1. Wstęp	7
1.1. Wprowadzenie	7
1.2. Cel pracy	7
1.3. Zawartość pracy	7
2. Budowa i rozwój programu	9
2.1. Specyfikacja wymagań	9
2.2. Wybór języka implementacji	9
2.2.1. Java a C i C++	9
2.2.2. Java a swobodne oprogramowanie	10
2.3. Definicje ważniejszych pojęć	10
2.4. Ważniejsze decyzje konstrukcyjne	11
2.4.1. Sposób działania	11
2.4.2. Podział na moduły	11
2.4.3. Interfejs użytkownika	12
2.4.4. Definiowanie testów kolokacji	12
2.5. Obecna struktura programu	13
2.5.1. Crawler	13
2.5.2. IndexBuilder	13
2.5.3. Archive	13
2.5.4. SAMain	14
2.6. Informacja zwrotna od użytkowników	17
2.6.1. Prezentacja na seminarium	17
2.6.2. Publiczny serwer zapytań	17
2.6.3. Wprowadzenie tablic sufiksowych	17
2.6.4. Umożliwienie selekcji wyników	17
2.7. Obecna funkcjonalność programu	17
3. Pozyskiwanie tekstów	19
3.1. Korpusy i ich zastosowania	19
3.1.1. Badanie tzw. uzusu	19
3.1.2. Łączliwość i kolokacje	19
3.1.3. Wybór synonimu	19
3.1.4. Wyszukiwanie słów kluczowych	20
3.2. Selekcja stron	20
3.2.1. Selekcja według witryny	20
3.2.2. Selekcja według działu witryny lub charakteru danych	20
3.2.3. Selekcja według rozszerzeń nazw plików	21

3.2.4.	Selekcja według języka	21
3.2.5.	Algorytm pelzania	21
3.3.	Konwersja do formatu archiwum	22
3.3.1.	Format archiwum	22
3.3.2.	Tłumaczenie encji	23
3.3.3.	Dodatkowa konwersja	23
3.3.4.	Kodowanie wejściowe i kodowanie archiwum	23
3.4.	Podział strony na elementy	23
3.4.1.	Ignorowalne znaczniki	24
3.4.2.	Ignorowalne elementy	24
3.5.	Selekcja istotnych elementów strony	24
3.5.1.	Odrzucanie krótkich elementów	24
3.5.2.	Odrzucanie powtarzających się elementów	25
3.5.3.	Studium przypadku – ogłoszenia na portalu MIMUW	25
3.6.	Ręczna korekta zawartości archiwum	26
3.6.1.	Usunięcie pojedynczych stron	26
3.6.2.	Usunięcie fragmentu strony	27
3.6.3.	Zmiany w segmentacji tekstu	27
3.6.4.	Segmentacja na zdania	27
3.6.5.	Korekty błędów filtrowania	27
3.6.6.	Korekta błędów źródeł	27
3.7.	Ostateczny efekt działania modułu	27
4.	Kolokacje	29
4.1.	Kolokacje, ich rodzaje i wykorzystanie	29
4.1.1.	Definicje	29
4.1.2.	Wykorzystanie kolokacji	30
4.2.	Automatyzacja wykrywania kolokacji	30
4.3.	Segmentacja tekstu i jego podział na słowa	31
4.3.1.	Podział elementów na segmenty	31
4.3.2.	Dualna natura kropki	32
4.3.3.	Podział segmentów na słowa	32
4.3.4.	Czy znaki rozgraniczające są słowami?	32
4.3.5.	Ignorowanie wybranych słów przy budowie indeksu	33
4.3.6.	Ignorowane słowa a definicja kolokacji	33
4.4.	Struktura indeksu	34
4.4.1.	Wystąpienia	35
4.4.2.	Flagi	35
4.4.3.	Nazwy własne	35
4.5.	Przegląd miar statystycznych służących do wykrywania kolokacji	35
4.5.1.	Częstość / liczba wystąpień	37
4.5.2.	t-score / z-score	37
4.5.3.	Test chi-kwadrat Pearsona	38
4.5.4.	t-score Studenta	38
4.5.5.	Porównanie prawdopodobieństw (LLR)	38
4.5.6.	Poprawione porównanie prawdopodobieństw	39
4.5.7.	Informacja wzajemna (MI)	39
4.5.8.	Przeskalowana informacja wzajemna (MMI)	39
4.5.9.	Pointwise Mutual Information (PMI)	39

4.5.10.	Symetryczne prawdopodobieństwo warunkowe (SCP)	40
4.5.11.	Wzór Dice'a	40
4.5.12.	Zależność wzajemna (MD)	40
4.5.13.	Log Frequency biased MD (LFMD)	40
4.5.14.	Frequency biased Symmetric Conditional Probability, FSCP	40
4.5.15.	Document Frequency	40
4.5.16.	Residual Inverse Document Frequency	41
4.5.17.	RIDF a pozostałe miary	41
4.6.	Metody filtrowania	41
4.6.1.	Odrzucanie kolokacji rzadkich	41
4.6.2.	Odrzucanie kolokacji pochodzących z jednego źródła	42
4.6.3.	Odrzucanie kolokacji częstych słów	42
4.6.4.	Odrzucanie kolokacji zawierających liczby	42
4.6.5.	Odrzucanie nazw własnych	42
4.7.	Porównanie efektywności metod	42
4.7.1.	Figure of Merit	42
5.	Możliwości rozbudowy programu	45
5.1.	Moduł Crawler	45
5.1.1.	Rozpoznawanie języka	45
5.1.2.	Odrzucanie elementów zawierających za dużo błędów	45
5.1.3.	Parser	46
5.1.4.	Podział plików tekstowych na elementy	46
5.2.	Indeks kolokacji	46
5.2.1.	Indeksowanie sąsiedztw kolokacji	46
5.3.	Miary statystyczne służące do wykrywania kolokacji	46
5.3.1.	Modyfikowanie istniejących miar	46
5.3.2.	Dodawanie nowych miar	48
5.3.3.	Usuwanie miar	48
5.3.4.	Miary nietypowe	48
5.4.	Filtrowanie danych wejściowych i wyników	48
5.4.1.	Odrzucanie kolokacji podstawialnych	48
5.4.2.	Sprowadzanie do formy podstawowej	49
5.5.	Kolokacje dłuższe niż dwa słowa	49
5.5.1.	Złożoność obliczeń	50
5.5.2.	Informacja wzajemna	50
5.5.3.	Uogólnianie przez szacowanie	50
5.6.	Obliczanie ilościowych wskaźników jakości testów	51
5.7.	Zastosowanie sieci neuronowej do uczenia programu	52
5.8.	Internacjonalizacja i lokalizacja programu	52
6.	Podsumowanie i wnioski	55
	Bibliografia	57
A.	Program Kolokacje	59

Rozdział 1

Wstęp

1.1. Wprowadzenie

Internet jest niezwykle bogatym i łatwo dostępnym źródłem tekstów. Ponadto teksty publikowane w Internecie znacznie szybciej odzwierciedlają nowe tendencje w języku niż publikowane metodami tradycyjnymi. Dlatego Internet może być dla lingwisty naturalnym źródłem żywego materiału badawczego, pozwalającego wnioskować o współczesnym stanie języka i zachodzących w nim procesach.

Wykorzystanie Internetu umożliwia skrócenie i uproszczenie drogi od napisania tekstu poprzez jego publikację po wykorzystanie w badaniach lingwistycznych. Praca niniejsza poświęcona jest automatyzacji drugiej części tej drogi, czyli przystosowywaniu typowych zbiorów tekstów dostępnych w Internecie – stron prywatnych i instytucji, serwisów prasowych, bibliotek – do potrzeb badań lingwistycznych.

1.2. Cel pracy

Pierwotnym celem pracy było opracowanie działającego programu – robota internetowego – indeksującego zasoby ze wskazanego podzbioru Internetu do potrzeb lingwistycznych, np. tworzenia korpusu lub słownika – i jego przetestowanie w praktyce na przykładzie pozyskiwania tekstów internetowych serwisów prasowych w wybranych językach. W trakcie pisania oprogramowania, w związku z zapotrzebowaniem zgłaszanym przez przyszłych użytkowników, zakres pracy rozszerzył się o przetwarzanie już pozyskanego tekstu. Ostatecznie ciężar pracy rozłożył się po równo pomiędzy samo pozyskiwanie zbioru tekstów z Internetu, a konkretny przykład jego wykorzystania – statystyczne metody wykrywania kolokacji.

1.3. Zawartość pracy

Oprócz niniejszego wstępu, praca składa się z pięciu rozdziałów oraz dodatku:

Rozdział drugi – *Budowa i rozwój programu* – prezentuje krótko dotychczasowy rozwój programu, jego obecną strukturę, podział na moduły oraz uzasadnienie podjętych decyzji konstrukcyjnych.

Rozdział trzeci – *Pozyskiwanie tekstów* – przedstawia problemy związane z wykorzystaniem zawartości Internetu do celów badań lingwistycznych, metody selekcji stron internetowych i fragmentów tych stron do tworzonego korpusu.

Rozdział czwarty – *Kolokacje* – poświęcony jest wykrywaniu kolokacji i opisuje zaimplementowane metody segmentacji, statystyczne miary siły kolokacji oraz metody filtrowania

wyników.

Rozdział piąty – *Możliwości rozwoju* – zawiera przegląd otwartych problemów oraz praktyczne wskazówki dla osób zainteresowanych rozwojem programu lub jego dostosowaniem do własnych potrzeb.

Rozdział szósty – *Podsumowanie i wnioski* – podsumowuje pracę i osiągnięte rezultaty.

Dodatek – *Program Kolokacje* – to płyta CD zawierająca program Kolokacje wraz ze źródłami, dokumentacją użytkownika, dokumentacją techniczną oraz przykładowymi archiwami tekstów.

Rozdział 2

Budowa i rozwój programu

2.1. Specyfikacja wymagań

Przyjęte podejście odbiegało od sztywnej, formalnej specyfikacji wymagań, właściwej dla oprogramowania przemysłowego. Starano się jak najszybciej uruchomić działający prototyp programu, zdefiniować możliwe kierunki rozwoju, a następnie stopniowo uzupełniać lub poprawiać funkcjonalność programu na podstawie uwag i propozycji zgłaszanych przez (potencjalnych) użytkowników.

Pierwotne oczekiwania zostały sformułowane jako:

1. Program ma służyć do pozyskiwania z Internetu tekstów do badań lingwistycznych (w tym tekstów internetowych serwisów prasowych w wybranych językach).
2. Program ma się podobać użytkownikom.

Po prezentacji pierwszych wersji programu, konsultacji z promotorem i uczestnikami seminarium zdecydowano się na kierunek rozwoju „wykrywanie kolokacji”. Do oczekiwań dodano:

3. Program ma, korzystając ze znanych testów statystycznych, wykrywać w pozyskanych tekstach kolokacje;
4. Program ma umożliwiać porównywanie wyników różnych testów statystycznych do wykrywania kolokacji.

2.2. Wybór języka implementacji

2.2.1. Java a C i C++

Jako język implementacji wybrana została Java. Zdecydowały o tym:

- wsparcie dla korzystania z Internetu;
- wsparcie dla przetwarzania tekstu;
- obsługa różnych kodowań;
- wsparcie dla wyrażeń regularnych;
- wsparcie dla I18N (internacjonalizacji);

- czytelność kodu, łatwość tworzenia dokumentacji technicznej;
- łatwość stworzenia działającego prototypu i stopniowego jego udoskonalania;
- przenośność i jednoznaczność¹ kodu.

Inne rozważane języki – C i C++ – oferowały większą szybkość wykonania programów. Jednak w pierwotnej fazie rozwoju programu – ukierunkowanej na pozyskiwanie tekstów z Internetu – szybkość wykonywania obliczeń przez program nie miała większego znaczenia (decydujące znaczenie dla czasu działania modułu Crawler ma szybkość połączenia z Internetem), a po podjęciu decyzji o kontynuacji pracy i dalszym rozwoju programu języki C/C++ nie dawały szansy na samodzielne ukończenie działającej, przenośnej i użytecznej aplikacji zamierzonych rozmiarów.

2.2.2. Java a swobodne oprogramowanie

Udostępniane przez Sun Microsystems Java Software Development Kit oraz Java Runtime Environment nie należą do swobodnego (wolnego) oprogramowania. Rozpowszechniane są bezpłatnie, na licencji, która daje prawo redystrybucji, ale zabrania modyfikowania klas z podstawowych pakietów. Kod źródłowy dołączony jest tylko w celach demonstracyjnych. Licencja zniechęca również do pisania w Javie oprogramowania dla elektrowni atomowych.²

Pojawiają się kompilatory Javy na licencji GPL (The GNU General Public License), nie są one jednak jeszcze w 100% zgodne ze specyfikacją Sun-a. Podejmowane były próby skompilowania i uruchomienia programu za pomocą `gcc` (The Gnu Compiler for Java), jednak dostępna wówczas wersja (`gcc 3.3.2`) nie implementowała jeszcze poprawnie niektórych często wykorzystywanych w programie funkcji, m.in. operacji na napisach z użyciem wyrażeń regularnych (np. `java.lang.String.matches`, `java.lang.String.replaceAll`, `java.lang.String.split`), definiowania kodowania strumienia (konstruktorzy `java.io.PrintStream`), a także konwersji URI/URL (np. `java.io.File.toURI`).

2.3. Definicje ważniejszych pojęć

Używane w niniejszej pracy pojęcia oznaczają:

Witryna – zbiór powiązanych ze sobą odsyłaczami stron internetowych, tworzących razem spójną logiczną całość.

Strona – dostępny w sieci dokument HTML lub XML.

Element – zawartość tekstowa pomiędzy kolejnymi znacznikami dokumentu HTML lub XML³

Segment – „wyrób zdaniopodobny” – ciąg słów stanowiący zdanie lub logicznie wydzielony fragment zdania (np. zdanie podrzędne, krótki cytat albo wtręt w nawiasie).

Słowo – ciąg znaków (liter, cyfr) ograniczony odstępami.

Pełzanie – proces poruszania się po sieci Internet przez odpowiedni program komputerowy (robot), rozpoczynający od wskazanej strony i rekurencyjnie udający się na strony wskazane w odsyłaczach na dotychczas odwiedzonych stronach.

¹Np. określenie w specyfikacji języka rozmiarów typu `int`.

²*Licensee acknowledges that Licensed Software is not designed or intended for use in the design, construction, operation or maintenance of any nuclear facility*

³Tak zdefiniowany na potrzeby niniejszej pracy element – nawet jeśli mowa jest o dokumentach XML – nie jest tożsamy z elementem XML, w skład którego wchodzi wszystko od znacznika początku elementu do znacznika końca elementu włącznie.

Również wyszukiwarka kolokacji w trakcie rozwoju programu uległa podziałowi na mniejsze moduły.

2.4.3. Interfejs użytkownika

Rozważane były trzy rozwiązania, jeśli chodzi o interfejs użytkownika:

1. Rozwiązanie „klasyczne” – przypominające typową wyszukiwarę internetową – interfejs PHP z serwerem zapytań;
2. Rozwiązanie „oszczędne” – zapytania zadawane są z konsoli, w odpowiedzi program generuje i zapisuje na dysku dokumenty HTML lub XML z wynikami;
3. Rozwiązanie oparte na komponentach Javy – program jest samodzielnym (*stand-alone*) apletem Javy.

Początkowo rozwijane były równolegle dwa pierwsze rozwiązania jako niezależne, alternatywne propozycje interfejsu. Jednak gdy program przekroczył pewien poziom złożoności, utrzymywanie różnych wersji interfejsu stawało się coraz bardziej pracochłonne. Należało się zdecydować na konkretne rozwiązanie. Wybrano połączenie rozwiązania „oszczędnego” i „klasycznego” – generowanie części stron statycznie, a pozostałych dynamicznie (poprzez PHP + serwer zapytań).

Jednak to podejście, zorientowane na sytuację „jeden korpus – wielu użytkowników” okazało się mało praktyczne dla użytkowników indywidualnie pracujących nad konkretnym zbiorem tekstów. Aby skrócić dla nich czas pomiędzy budową indeksu a otrzymaniem wyników, do ostatecznej wersji programu dodano też moduł umożliwiający zarządzanie archiwum poprzez samodzielny aplet Javy.

2.4.4. Definiowanie testów kolokacji

Program z założenia miał umożliwiać testowanie różnych miar statystycznych służących do wykrywania kolokacji. Dlatego ważne było zaprojektowanie dobrego sposobu definiowania tych miar, tak by można było je łatwo dodawać, modyfikować lub usuwać.

Początkowo rozważany był pomysł definiowania testów za pomocą specjalnej składni interpretowanej przez program, jednak ostatecznie wybrano definiowanie miar bezpośrednio w Javie. Każda miara stanowi odrębną klasę, podklasę abstrakcyjnej klasy `CollocationTest`.

Wadą tego rozwiązania jest to, że do definiowania nowych lub modyfikowania istniejących testów potrzebny jest kompilator Javy. Na szczęście – przynajmniej w przypadku kompilowania do kodu JVM (Java Virtual Machine) – nie jest konieczne ponowne linkowanie programu.

Do zalet należy:

- Łatwość implementacji mechanizmu;
- Możliwość korzystania przy definiowaniu testów ze wszystkich funkcji modułu `Math`;
- Możliwość korzystania przy definiowaniu testów ze zmiennych, funkcji wejścia/wyjścia, hierarchii klas i innych możliwości języka Java;
- Sprawdzanie poprawności definicji testu przez wbudowany parser;
- Większa efektywność, dzięki częściowej optymalizacji kodu podczas kompilacji.

2.5. Obecna struktura programu

Program składa się z trzech głównych modułów, odpowiadającym kolejnym etapom pracy nad pozyskiwanym z internetu tekstem – **Crawler**, **IndexBuilder** oraz **Archive**. Ostatni z nich występuje w trzech wersjach, odpowiadających różnym wariantom interfejsu użytkownika.

2.5.1. Crawler

Pierwszy z modułów odpowiada za przeszukiwanie Internetu, selekcję stron, selekcję elementów strony WWW, ich konwersję i zapis w jednolitym formacie na dysku lokalnym. Pracę tego modułu można w dowolnym momencie przerwać, a następnie wznowić w tym samym punkcie. Można też ściągać do archiwum wiele wersji tej samej strony (np. codzienny serwis prasowy).

2.5.2. IndexBuilder

Drugi etap pracy programu to budowa indeksu kolokacji. Moduł wczytuje z dysku lokalnego wcześniej ściągnięte i skonwertowane strony, a następnie uzupełnia katalog archiwum o pliki indeksu, umożliwiające szybki dostęp zarówno do danych statystycznych, jak i wyszukiwanie konkretnych słów lub kolokacji.

2.5.3. Archive

Kolejnym etapem jest udostępnienie danych zawartych w archiwum w formie wygodnej dla użytkownika. Aby osiągnąć większą efektywność, moduł podzielony został na dwie części (podklasy klasy Archive – **PrettyPrinter** i **QueryServer**). Trzecia podklasa – **SAManager** – to alternatywna propozycja interfejsu użytkownika.

PrettyPrinter

Moduł **PrettyPrinter** generuje statyczne strony HTML udostępniające najważniejsze statystyki archiwum – spis zindeksowanych stron, listę najczęściej występujących słów, rankingi najsilniejszych kolokacji według różnych miar, a także porównanie tych rankingów. Możliwe jest również wygenerowanie za pomocą tego modułu rankingu kolokacji konkretnego słowa.

QueryServer

Moduł **QueryServer** służy interaktywnemu udostępnianiu zasobów archiwum. Umożliwia zadanie zapytań o kolokacje zadanego słowa, o przykłady (konteksty) wystąpień zadanego słowa lub kolokacji, a także o słowa zawierające zadany rdzeń. Moduł może przyjmować zapytania na porcie zewnętrznym albo bezpośrednio z konsoli.

Interfejs WWW

Najlepszy rezultat przynosi wykorzystanie obu wyżej wymienionych modułów archiwum. **PrettyPrinter** generuje statyczne strony WWW (czysty HTML), zawierające odwołania do stron dynamicznych (PHP) umożliwiających zadanie bardziej szczegółowych zapytań, obsługiwanych następnie przez **QueryServer**. Np. kliknięcie na jedną z kolokacji w wygenerowanym statycznie ogólnym rankingu lub porównaniu rankingów powoduje przejście do generowanej

dynamicznie listy przykładowych kontekstów tej kolokacji, a kliknięcie na słowo w wygenerowanym statycznie słowniku frekwencyjnym – przejście do generowanego dynamicznie rankingu kolokacji dla tego słowa.

Skąd ten podział?

PrettyPrinter zajmuje się przygotowaniem danych wymagających przejrzania całego archiwum, dlatego złożoność większości z algorytmów wykorzystywanych przez ten moduł wynosi $O(n \log n)$. Ponieważ n jest rzędu przynajmniej setek tysięcy, wielokrotne zadawanie tego typu zapytań w trybie interaktywnym byłoby dużym obciążeniem dla serwera. Wprawdzie przy pracy samodzielnej nie ma to większego znaczenia, ale w przypadku, w którym z zestawień korzysta większa ilość osób, wygenerowanie i zachowanie statycznych stron HTML pozwala na uniknięcie wielokrotnego wykonywania tej samej pracy.

Z drugiej strony, nie jest sensowne generowanie statycznych stron HTML dla wszystkich możliwych zapytań (np. o konteksty wystąpień dla kilkudziesięciu tysięcy różnych par słów). Dobrze jest też umożliwić użytkownikowi pewną swobodę np. w sprawie długości kontekstów, czy preferowanej miary siły kolokacji. Dlatego powstał również moduł QueryServer.

Zapytania obsługiwane przez QueryServer mają złożoność znacznie niższą od tych obsługiwanych przez PrettyPrinter:

- Zapytanie o kolokacje słowa w : $O(\log n k(w))$
- Zapytanie o konteksty wystąpień słowa lub kolokacji w : $O(\log n c(w) s)$
- Zapytanie o słowa zawierające dany rdzeń, prefiks lub sufiks: średnio $O(\log n)$, pesymistycznie n (np. liczba słów zawierających „a” jest porównywalna z liczbą wszystkich słów) – dlatego tutaj zostało zastosowane dodatkowe ograniczenie co do liczby wyników.

Oznaczenia: n – rozmiar korpusu (liczba słów); $c(w)$ – liczba wystąpień słowa w ($c(w) \ll n$); $k(w)$ – liczba słów tworzących kolokacje z w ($k(w) \leq c(w)$); s – długość kontekstu (z reguły rzędu 100 znaków).

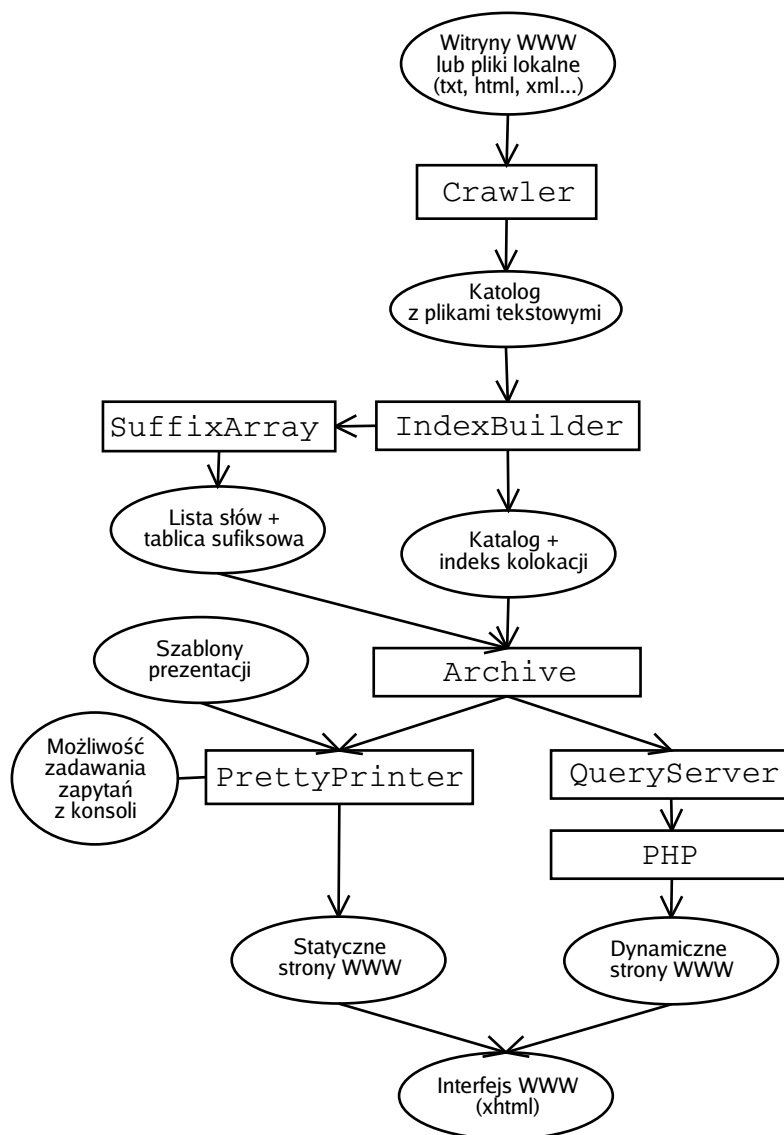
SAManager

Moduł SAManager to alternatywne podejście do konstrukcji interfejsu użytkownika, zorientowane na użytkowników indywidualnie pracujących nad zbiorem tekstów, oparty na komponentach Javy. Umożliwia skorzystanie z podstawowych funkcji modułu za pomocą kilku kliknięć myszką.

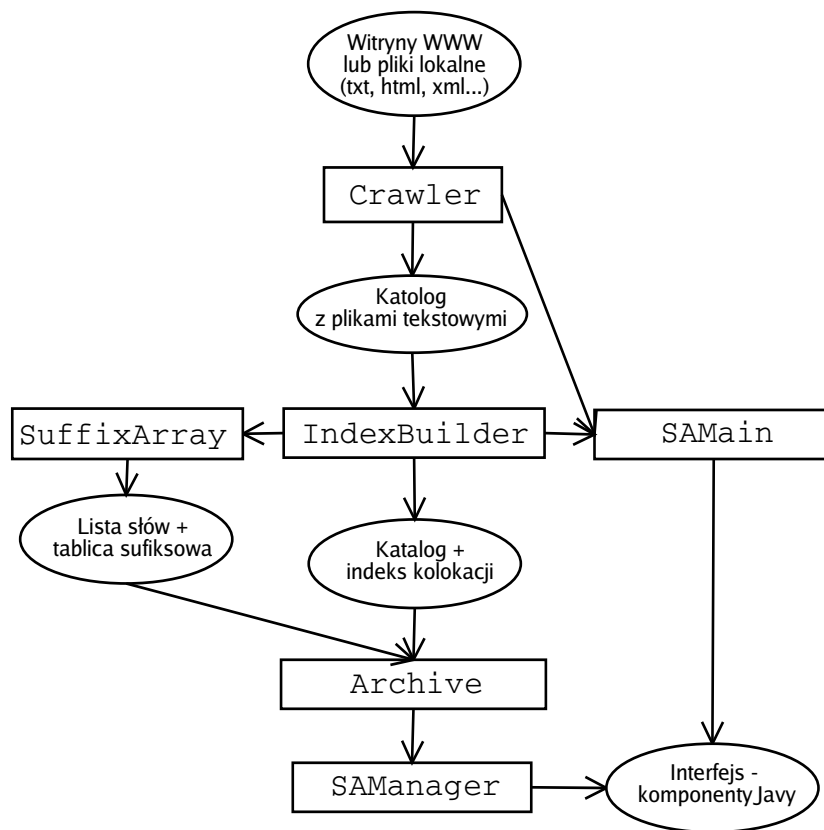
2.5.4. SAMain

SAMain to rozwinięcie pomysłu zawartego w module SAManager – zorientowany na indywidualnego użytkownika graficzny interfejs umożliwiający prosty dostęp do podstawowych funkcji wszystkich modułów programu.

Oprócz udostępnienia graficznego interfejsu, SAMain pozwala pominąć pewne etapy nie konieczne przy pracy indywidualnej (np. zapisywanie i odczytywanie indeksu), co przydaje się np. przy testowaniu wpływu różnych konfiguracji modułu IndexBuilder na wyniki konkretnego zapytania.



Rysunek 2.2: Przepływ danych pomiędzy modułami programu w wersji przeznaczonej do udostępniania wyników przez Internet („server”)



Rysunek 2.3: Przepływ danych pomiędzy modułami programu przy pracy samodzielnej („standalone”)

2.6. Informacja zwrotna od użytkowników

2.6.1. Prezentacja na seminarium

Od początku dążono do jak najszybszego uruchomienia i udostępnienia potencjalnym użytkownikom działającego programu (być może w początkowej fazie z pewnymi uproszczeniami, niepełną funkcjonalnością i nieoptymalnymi algorytmami). Celem tego było zebranie informacji zwrotnej od użytkowników, doprecyzowanie potrzeb, wymagań i preferencji.

Poszczególne stabilne wersje programu (0.2, 0.5 i 0.6) były prezentowane na seminariach „Narzędzia i metody przetwarzania tekstów” oraz „Lingwistyka informatyczna i przetwarzanie tekstów” w Instytucie Informatyki, Instytucie Orientalistycznym oraz Katedrze Lingwistyki Formalnej.

2.6.2. Publiczny serwer zapytań

W październiku 2003 r. uruchomiony został publicznie dostępny serwer zapytań z interfejsem WWW dla trzech przykładowych archiwów:

- Witryna Rady Języka Polskiego (język polski, ok. 45 tys. słów)
- Dokumentacja Emacsa (język angielski, ok. 225 tys. słów)
- Wiadomości Deutsche Welle (język suahili, ok. 160 tys. słów)

Zbierane uwagi były stopniowo uwzględniane w kolejnych wersjach programu. W efekcie usunięto sporo błędów, a do podstawowej funkcjonalności programu dodano m.in. słownik frekwencyjny oraz tablice sufiksowe.

2.6.3. Wprowadzenie tablic sufiksowych

Tablice sufiksowe umożliwiły wyszukiwanie słów na podstawie ich rdzenia, prefiksu lub sufiksu. W wersji 0.6 rozszerzono ich wykorzystanie na wyszukiwanie kolokacji oraz kontekstów (tak, by można było zestawzić w jednej tabeli kolokacje np. dla **wyraz***, czyli wszystkich słów rozpoczynających się od *wyraz*). Kolejnym logicznym rozszerzeniem było dopuszczenie w zapytaniach alternatywy (tak by można było np. zapytać o kolokacje dla **man men**).

2.6.4. Umożliwienie selekcji wyników

Również uwagi użytkowników były przyczyną rozbudowania interfejsu selekcji wyników. Dodano możliwości uwzględniania bądź nie kolokacji lewo- i prawostronnych, nazw własnych, kolokacji z liczbami.

2.7. Obecna funkcjonalność programu

Obecnie program potrafi:

- pobierać strony ze wskazanych witryn, podążając za odsyłaczami tylko wtedy, gdy nie wyprowadzają one poza zadany zbiór witryn;
- przerywać i wznowiać pobieranie, a także dodawać nowe witryny do istniejącego zbioru tekstów, albo systematycznie indeksować nowości na witrynie zmiennej w czasie;

- filtrować zawartość stron, nie kopiując elementów powtarzających się lub o znaczeniu dekoracyjnym;
- zbudować dla ściągniętych stron indeks wystąpień, umożliwiający szybkie znajdowanie słów, kolokacji oraz kontekstów ich wystąpień;
- stworzyć słownik frekwencyjny dla ściągniętych tekstów;
- tworzyć zestawienia słów zawierających dany rdzeń, prefiks lub sufiks;
- sortować zestawienia słów alfabetycznie, według liczby wystąpień lub w porządku *a tergo*, uwzględniając przy tym różnice w porządkach alfabetycznych różnych języków;
- obliczać wartości 14 testów statystycznych, służących do wykrywania kolokacji;
- tworzyć zestawienia wyników według konkretnego testu lub porównania wyników różnych testów;
- tworzyć ww. zestawienia dla pojedynczych słów, grup słów lub całego korpusu;
- tworzyć zestawienia przykładowych kontekstów dla słów lub kolokacji;
- obsługiwać zapytania zadawane przez interfejs WWW lub przez samodzielną aplikację;
- zapisywać wyniki zapytań w postaci plików tekstowych, HTML lub CSV.

Rozdział 3

Pozyskiwanie tekstów

Rozdział ten przedstawia problemy związane z wykorzystaniem zawartości Internetu do celów badań lingwistycznych. Zawiera propozycje metod selekcji stron internetowych i fragmentów tych stron do tworzonego korpusu.

3.1. Korpusy i ich zastosowania

Celem modułu **Crawler** jest stworzenie korpusu tekstów. Korpus – według Polskiego Wydawnictwa Naukowego PWN [PWN] – to dowolny zbiór tekstów, w którym czegoś szukamy. Dla lingwisty korpus to autentyczny materiał językowy, na którego podstawie wnioskuje się o znaczeniu wyrazów i połączeń wyrazowych. Postaramy się nieco doprecyzować tę definicję, krótko prezentując przykłady zastosowań korpusu, gdyż zakres zastosowań ma znaczenie dla konstrukcji i sposobu działania modułu.

3.1.1. Badanie tzw. uzusu

Sprawdzenie częstości i rozrzutu występowania słowa pomiędzy różnymi dokumentami pozwala zbadać, czy dane słowo przyjęło się do powszechnego stosowania w danym języku.

W przypadku słów zapożyczonych z innego języka, o jeszcze nieugruntowanej pisowni, można sprawdzić, która transkrypcja jest najbardziej popularna i przyjąć ją za najbardziej poprawną (np. *biznesman* vs *biznesmen* albo *lobbing* vs *lobbying*).

3.1.2. Łączliwość i kolokacje

Łączliwość to skłonność danych słów do występowania w tekstach obok siebie. Łączliwość obejmuje zarówno sztywne wymagania składniowe, jak i bardziej nieostre zjawiska, wynikające z naszych przyzwyczajzeń i skojarzeń. Często występujące obok siebie słowa określamy mianem kolokacji – i o tym traktuje rozdział czwarty.

3.1.3. Wybór synonimu

Tłumacze często stają przed problemem wyboru jednego z kilku możliwych synonimów jako odpowiednika przekładanego słowa. Sprawdzenie kontekstów wystąpień lub kolokacji tego słowa ułatwia dokonanie właściwego wyboru.

3.1.4. Wyszukiwanie słów kluczowych

Badając rozrzut występowania słów pomiędzy różnymi dokumentami w dobrze określonym zbiorze tekstów, można wytypować słowa lub ciągi słów dobrze nadające się na słowa kluczowe, pozwalające podzielić zasoby na mniejsze dziedziny. Podobnych metod można też użyć np. do wybierania haseł do indeksu alfabetycznego.

3.2. Selekcja stron

Przy tworzeniu korpusu tekstów pozyskanych z Internetu interesują nas z reguły strony wybierane według pewnego klucza, np. strony w konkretnym języku, strony z określonej witryny lub zbioru witryn.

3.2.1. Selekcja według witryny

Przez witrynę internetową (ang. *website*) rozumiemy zbiór powiązanych ze sobą stron internetowych, tworzących spójną logiczną całość.

Pytanie, z którym musi się zmierzyć program, brzmi „jak rozstrzygnąć czy strona X należy do tej samej witryny co strona Y?”.

Najprostsze wyróżnienie to wspólny prefiks w URL, np. adresy wszystkich stron z witryny Życia Warszawy zaczynają się od `http://www.zw.com.pl/`, a z witryny Jana Kowalskiego – od `http://jankowalski.webpark.pl/`. Niestety, to nie wystarcza – niektóre witryny (np. Gazety Wyborczej) podzielone są pomiędzy kilka serwerów.

Na potrzeby tworzonego programu przyjęto identyfikację witryny poprzez wyrażenie regularne opisujące adresy stron składających się na witrynę. Rozwiązanie to wiąże się z koniecznością chociaż pobieżnego orientowania się przez użytkownika w strukturze witryny, którą chce zbadać. Ponieważ jednak jest to czynność wykonywana stosunkowo rzadko, jej automatyzacja nie wydaje się celowa.

Uwaga: program po zadaniu punktu startowego porusza się po witrynie za pomocą znajdujący się na stronach odsyłaczy. Możliwa zatem jest sytuacja, w której strona o URL pasującym do zadanego wyrażenia regularnego nie zostanie zindeksowana, jeżeli brak będzie możliwości dojścia do niej z punktu startowego bez wychodzenia poza witrynę.

3.2.2. Selekcja według działu witryny lub charakteru danych

Uboczną zaletą obranego podejścia jest możliwość wykorzystania wyrażeń regularnych do wprowadzenia bardziej złożonych kryteriów doboru stron do zindeksowania. Z reguły treść URL odzwierciedla w mniejszym lub większym stopniu zawartość danej strony i funkcję jaką pełni ona na danej witrynie.

Na przykład wyrażenie regularne:

```
http://wiadomosci.poprostu.pl/(?id=[0-9]+)?(&row=[0-9]+)?
```

– opisuje aktualne wiadomości (bez komentarzy czytelników, dłuższych felietonów i wiadomości archiwalnych).

Selekcja może obejmować parę witryn lub ich działów. Jeżeli na przykład chcemy pobrać polską witrynę GNU oraz teksty po polsku na międzynarodowej witrynie GNU, właściwy plik konfiguracyjny powinien zawierać dwie linijki:

```
http://www.gnu.org.pl/.*
```

```
http://www.gnu.org/.*\.pl.html
```

3.2.3. Selekcja według rozszerzeń nazw plików

Specjalnym przypadkiem selekcji według charakteru danych jest selekcja według rozszerzeń ściąganych stron. Ze względu na to, że dokonywanie tej selekcji w ramach wyrażenia regularnego opisującego witrynę byłoby skomplikowane i zaciemniałoby obraz sytuacji, stworzono dodatkowy test, a lista dopuszczalnych rozszerzeń została wydzielona do odrębnego pliku konfiguracyjnego.

Domyślnie program ściąga wszystkie pliki z rozszerzeniem `.htm`, `.html`, `.xhtml`, `.xml`, `.txt`, `.php`, `.asp`.

Obecna wersja programu nie różnicuje swojego zachowania w zależności od rodzaju pliku, tzn. np. rozpoznając język sprawdza te same znaczniki, niezależnie od tego czy ma do czynienia z dokumentem HTML czy XHTML.

3.2.4. Selekcja według języka

Język zawartości tekstowej strony WWW opisywany jest z reguły na jeden z dwóch sposobów:¹

- jako atrybut znacznika HTML, np. `<html lang="pl">`
- znacznikiem meta, np. `<meta http-equiv="Content-Language" content="pl">`

Obecnie program sprawdza oba te znaczniki. Jeśli włączona jest selekcja według języka, to w zależności od konfiguracji:

- strona jest pobierana wtedy i tylko wtedy, gdy nie występuje na niej żaden znacznik określający język inny niż pożądanym;
- strona jest pobierana wtedy i tylko wtedy, gdy występuje na niej choć jeden znacznik określający pożądanym język.

W praktyce jednak często autorzy stron w ogóle nie opisują języka strony. Dlatego w przyszłości – przy zastosowaniach programu wykraczających poza jedną lub kilka witryn – można spróbować uzupełnić procedurę rozpoznawania języka o proste testy zawartości treści strony, np. sprawdzanie czy na stronie występują popularne w danym języku słowa.

3.2.5. Algorytm pełzania

Można sobie wyobrazić sytuację, w której użytkownikowi nie zależy na archiwizowaniu całej witryny, ale jedynie na uzyskaniu losowej próbki tekstów w danym języku. Można też sobie wyobrazić, że użytkownikowi może w ogóle nie zależeć na konkretnej witrynie. W takich sytuacjach przydatna może się okazać możliwość:

- ustalenia pożądanego rozmiaru próbki
- wpływania na kolejność w jakiej wybierane są strony do zindeksowania

¹Teoretycznie atrybut `lang` może występować także przy innych znacznikach, określając część strony napisaną w innym języku. Można też zastosować specjalny znacznik `<lang>`. Autorowi nie udało się jednak znaleźć praktycznego przykładu – poza samouczkami HTML – zastosowania wyżej wymienionych.

Rozmiar próbki

Obecnie na rozmiar próbki można wpływać, ustalając maksymalną liczbę stron do zarchiwizowania.

Wydaje się, że pożądana byłaby możliwość określania rozmiaru próbki w słowach (najczęściej stosowana miara rozmiaru korpusu). Jednak segmentacja na słowa nie jest jednoznaczna – zależy od zdefiniowanego zestawu odstępów. Rozwijając program, zdecydowano się na dokonywanie segmentacji na słowa dopiero podczas budowy indeksu, tak by umożliwić łatwe testowanie alternatywnych segmentacji utworzonego korpusu. Dlatego nie jest możliwe określanie ilości słów w próbce już na etapie ściągania. Nie powinno to jednak być dużą niedogodnością, gdyż jeżeli okaże się, że liczba słów nie odpowiada oczekiwaniom użytkownika, zawsze możliwe jest wznowienie pełzania lub usunięcie części stron z korpusu.

Kolejność stron

Możliwe są trzy podstawowe rozwiązania:

- kolejka FIFO – odpowiada to przeszukiwaniu witryn „wszerz” (algorytm BFS) i „od lewej”
- kolejka LIFO (stos) – odpowiada to przeszukiwaniu witryn „w głąb” (algorytm DFS) i „od prawej”
- wybór losowy

Domyślnie stosowana jest kolejka LIFO, jako najprostsza i najbardziej efektywna przy indeksowaniu całych witryn (zaimplementowana jako tablica z możliwością zmiany rozmiaru).

Jednak jeśli zakładamy, że rozmiar próbki będzie mniejszy niż rozmiar witryny (witryn), to odpowiedniejszy będzie wybór losowy.

Z kolei zaletą zastosowania kolejki FIFO jest odbierany przez użytkownika jako bardziej „naturalny” porządek stron w archiwum. Być może w przyszłych wersjach programu warto dodać opcję wyszukiwania odsyłaczy na stronie od „końca”, tak by przy kolejności LIFO osiągnąć efekt przechodzenia witryn „w głąb” i „od lewej”.

3.3. Konwersja do formatu archiwum

3.3.1. Format archiwum

Podstawowym dylematem do rozstrzygnięcia przy konstruowaniu archiwum, było czy archiwizować stronę „tak jak jest”, z całym dobrodziejstwem HTML-owego inwentarza (ciężka, często nieczytelna, wymaga dalszej obróbki przed wykorzystaniem), czy skonwertować na czysty tekst (tracona jest informacja o strukturze)? A może istnieje jakaś trzecia droga?

Po dyskusji z przyszłymi użytkownikami przyjęte zostało rozwiązanie z konwersją na czysty tekst. Celem programu nie jest *mirrorowanie* witryn WWW (do tego świetnie nadaje się np. `wget`), ale pozyskiwanie próbek tekstu i ich dostarczanie innym programom (modułom programu) w jednolitej, łatwej do wczytania formie.

Ślad po strukturze HTML pozostaje w postaci segmentacji tekstu na wiersze – każdy *istotny* element (patrz niżej) zapisywany jest w osobnym wierszu.

3.3.2. Tłumaczenie encji

Konwersja HTML / XHTML do czystego tekstu wiąże się zazwyczaj z koniecznością przetłumaczenia encji, usunięcia komentarzy i znaczników oraz zmiany kodowania.

Moduł tłumaczy encje nazwane (np. `ä` czy `Ó`), numeryczne dziesiętne (np. `„`) oraz numeryczne heksadecymalne (np. `ç`).

Znaczenie encji nazwanych definiowane jest w pliku konfiguracyjnym `entities.ini` – dla każdej encji podany jest w kodzie szesnastkowym numer odpowiadającej jej znaku w Unikodzie.

3.3.3. Dodatkowa konwersja

W trakcie testowania programu, aby ułatwić przetwarzanie i segmentację tekstów, zdecydowano się wprowadzić pewne uproszczenia, ujednolicające zawartość archiwum. W szczególności:

- twarda spacja zastępowana jest zwykłą spacją;
- znaki tabulacji zastępowane są zwykłą spacją;
- ciągi następujących po sobie spacji zastępowane są pojedynczą spacją;
- apostrofy ukośne zastępowane są apostrofami prostymi;
- różne rodzaje cudzysłowów (znaki o numerach 8220-8222) zastępowane są cudzysłowami z zestawu znaków ASCII;
- „TeX-owe” reprezentacje cudzysłowów (za pomocą przecinków lub apostrofów) zastępowane są cudzysłowami z zestawu znaków ASCII.

3.3.4. Kodowanie wejściowe i kodowanie archiwum

Często zdarza się, że strona WWW nie zawiera informacji o kodowaniu, albo – co gorsza – zawiera informację błędną. Dlatego dodana została możliwość „wymuszania” konkretnego kodowania wejściowego.

Przyjęto założenie, że całe archiwum należy sprowadzić do jednolitego kodowania, tak by ułatwić zadanie późniejszym programom na nim operującym. Domyślnym kodowaniem archiwum jest UTF-8, jako kodowanie uniwersalne, a jednocześnie dość efektywne w przypadku wielu języków europejskich. Zrozumiałe jest jednak, że np. w przypadku tworzenia korpusu tekstów ukraińskich właściwsze może być KOI8U, dlatego kodowanie jest elementem konfigurowalnym.

3.4. Podział strony na elementy

Podział strony na elementy istotny jest dla modułu `IndexBuilder`, ale również dla selekcji jej istotnych fragmentów.

Na potrzeby niniejszego tekstu przyjmujemy, że elementem jest zawartość tekstowa pomiędzy kolejnymi *istotnymi* znacznikami dokumentu XML lub HTML. To bardzo praktyczna dla programu i łatwa do wyodrębnienia część strony.

Uwaga: tak zdefiniowany element – nawet jeżeli mowa jest o dokumentach XML – nie jest tożsamy z elementem XML, w skład którego wchodzi wszystko od znacznika początku elementu do znacznika końca elementu z samymi znacznikami włącznie.

3.4.1. Ignorowalne znaczniki

Przez ignorowalne znaczniki rozumiemy takie, o których możemy z dużym prawdopodobieństwem powiedzieć, że nie mają znaczenia dla logicznej struktury strony. O ile jednak w przypadku znacznika HTML `` rozstrzygnięcie jest raczej jednoznaczne, to już np. znaczniki `` czy `<i>` bywają używane do wyróżnienia tytułu albo cytatu (często zresztą w połączeniu ze znacznikiem ``).

To już niestety jest „wada wrodzona” HTML, gdzie znaczniki służą głównie prezentacji tekstu, a nie odwzorowaniu jego logicznej struktury. Wprawdzie niektóre nowsze strony, korzystające z arkuszy stylów, starają się rozdzielić warstwę struktury od warstwy prezentacji, ale nie można na taki podział liczyć pozyskując teksty z obecnych zasobów Internetu. Jednak przy badaniu konkretnej witryny można poświęcić chwilę na analizę źródeł stron, by poprzez prawidłowe zidentyfikowanie ignorowalnych znaczników poprawić jakość segmentacji na elementy.

3.4.2. Ignorowalne elementy

Za ignorowalne elementy uznajemy takie, których zawartość tekstowa nie powinna zostać zaliczona do zawartości tekstowej strony. Przykładem znacznika takiego elementu jest `<script>` – webmasterzy nie zawsze dbają o rekomendowane dodatkowe oznaczenie tej zawartości jako komentarza HTML.

3.5. Selekcja istotnych elementów strony

Korpus tekstów utworzony poprzez automatyczne ściąganie całych zawartości witryn będzie istotnie różnił się od korpusów utworzonych w tradycyjny sposób. Na typowej witrynie internetowej występuje wiele segmentów powtarzających się na każdej stronie witryny – np. menu, elementy nawigacyjne, informacja o prawach autorskich. Poza tym często występują też pojedyncze elementy o innym charakterze niż reszta strony – np. odsyłacze do innych wersji językowych będą w innym języku niż reszta strony.

Pojawia się pytanie – czy należy starać się jakoś zniwelować tego typu zaburzenia, czy wręcz przeciwnie, należy je zachować i zaakceptować jako specyfikę korpusu internetowego?

Autor preferuje ten pierwszy pogląd – kopiowanie stron WWW w całości odpowiadałoby np. włączaniu do tradycyjnego korpusu spisów treści, a także stopek i nagłówków występujących na każdej stronie książki czy czasopisma. Dlatego do programu zostały dodane opcje pozwalające na częściowe zniwelowanie tego zaburzenia.

3.5.1. Odrzucanie krótkich elementów

Jedną z charakterystycznych cech elementów menu czy odsyłaczy jest ich niewielka długość. Można zatem spróbować odrzucać elementy, których długość zawartości tekstowej, liczona w znakach lub słowach, jest krótsza od ustalonej wartości.

Takie rozwiązanie powinno sprawdzać się szczególnie dobrze, jeśli próbujemy pozyskać tekst na potrzeby badań, w których istotny jest kontekst, w jakim występuje słowo – np. tworzenia konkordancji albo wykrywania kolokacji. Przy takich badaniach element jednowyrazowy będzie praktycznie bezużyteczny, a najbardziej interesujące będą całe akapity jednorodnego tekstu.

Pojawia się tu jednak zagrożenie: możliwe jest wyrzucenie pojedynczych słów wyróżnionych odrębnym elementem – np. pogrubieniem albo odsyłaczem – z większego i stanowiącego logiczną całość tekstu. W szczególności bezpośrednia aplikacja tej metody filtrowania

nie sprawdza się przy pozyskiwaniu stron WWW stworzonych w niektórych edytorach typu WYSIWYG lub w wyniku konwersji dokumentów MSWorda.

Częściowo problem ten rozwiązuje dobre zdefiniowanie zestawu ignorowalnych znaczników elementów (patrz 3.4.1). Wymaga to jednak pewnej wiedzy o strukturze źródła strony i nie sprawdza się na stronach, na których te same znaczniki pełnią bardzo różne funkcje.

3.5.2. Odrzucanie powtarzających się elementów

Skoro chcemy się pozbyć powtarzających się segmentów, logiczne wydaje się niearchiwizowanie powtarzających się elementów. Można zatem tworzyć w trakcie indeksowania słownik już zindeksowanych elementów i ignorować elementy już znajdujące się w słowniku. Dzięki temu elementy np. menu pojawiają się w tekście korpusu raz i dokładnie raz.

Należy jednak zwrócić uwagę, że tu również – choć w mniejszym stopniu – pojawia się zagrożenie możliwością wyrzucenia z większego tekstu pojedynczych słów. W znakomitej większości przypadków filtr działa poprawnie, znacząco zwiększając stosunek sygnału do szumu.

Ze względów praktycznych (oszczędność pamięci i miejsca na dysku) w słowniku pamiętany jest nie cały element (często kilka tysięcy znaków), ale tylko jego długość i pierwszych kilkadziesiąt (domyślnie 40) znaków. Nie zaobserwowano przypadków błędnego odrzucania elementów wynikłego z tego przybliżenia. W razie potrzeby procedura może zostać łatwo zmodyfikowana, tak by w słowniku zapamiętywana była np. suma kontrolna elementu.

Ten sam mechanizm sprawdza się w przypadku wielokrotnego indeksowania tej samej, ale zmiennej w czasie witryny internetowej. Zapamiętanie słownika elementów pomiędzy kolejnymi wywołaniami programu pozwala poprawnie rozpoznać nowe i stare strony (a także zmienione fragmenty stron) wchodzące w skład witryny.

Co więcej – mechanizm jest odporny nawet na zmiany nazw stron źródłowych i przenosiny materiałów pomiędzy stronami. Jeśli np. jakieś informacje, w związku z tym że z bieżących staną się archiwalne, zostaną przeniesione ze strony `news.html` na stronę `archive.html`, można się spodziewać, że program nie będzie kopiował zawartości `archive.html`, a jedynie nowe fragmenty `news.html`.

3.5.3. Studium przypadku – ogłoszenia na portalu MIMUW

Przeanalizujemy działanie programu i dwóch opisanych wyżej filtrów na części portalu Wydziału Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego.

Witryna: [http://www.mimuw.edu.pl/news/ogloszenia/.*](http://www.mimuw.edu.pl/news/ogloszenia/)

Stan na dzień 14 grudnia 2003 r.

Dla każdego portalowego ogłoszenia strona dostarczana przez serwer ma rozmiar przynajmniej 13-16 kB (bez rysunków). Sama zawartość tekstowa takiej strony to ok. 3-5 Kb znaków. Treść ogłoszenia – czyli to, co jest na stronie unikalne i najbardziej interesujące – to z reguły zaledwie kilka zdań, od 100 do 1 kB znaków. W ekstremalnym przypadku (dyskusja na temat zajęć ogólnouniwersyteckich) treść ogłoszenia zawiera niecałe 10 Kb znaków (przy odpowiednio większych rozmiarach strony – 29 kB).

Na przykład dołączana do każdej strony wydziałowego portalu informacja, że:

Zamieszczone w portalu dokumenty nierzadko są wynikiem wielokrotnego transferu oryginalnych (źródłowych) dokumentów za pomocą różnych mediów, automatycznej transformacji lub konwersji na inne formaty, a także ich ręcznej edycji. Z tego też względu zamieszczone w portalu dokumenty nie mogą być traktowane jako równoważne dokumentom oryginalnym (źródłowym). Ponadto, nieunikniona ludzka omyłność może mieć także wpływ na ostateczną

treść, aktualność, bądź umiejscowienie dokumentów w portalu. Jakkolwiek MIMUW Webmaster i osoby z nim współpracujące dokładają wszelkich starań, aby zapewnić maksymalną aktualność i zgodność dokumentów zamieszczonych w portalu i ich oryginalnych odpowiedników, Wydział Matematyki, Informatyki i Mechaniki lub jakakolwiek związana z nim osoba, nie ponoszą żadnej odpowiedzialności za jakiegokolwiek skutki wynikłe z korzystania portalu, w tym – także za skutki wykorzystania lub niewykorzystania informacji w nim zamieszczonej.

– jest niewątpliwie cenna dla użytkownika portalu, ale wydaje się, że umieszczenie jej w korpusie kilkaset razy (kilka tysięcy w przypadku całego portalu) nie wzbogaca istotnie jego zawartości, a wręcz przeciwnie – może, poprzez znaczące zaburzenie statystyki, doprowadzić do wyciągnięcia fałszywych wniosków na temat słów i kolokacji charakterystycznych dla przytoczonej wyżej krótkiej informacji.

Uruchomiony z włączonym filtrem duplikatów program poprawnie rozpoznał powtarzające się elementy i skopiował do archiwum jedynie same ogłoszenia. Cała otoczka – nagłówek, odsyłacze do pozostałych wiadomości, innych działów portalu, dział „Polecamy”, informacje o pełnym wyłączeniu odpowiedzialności itp. – zarchiwizowana została dokładnie jeden raz. Poprawnie zostały zignorowane powtarzające się na odrębnych stronach informacje o dokumentach chronionych, braku uprawnień, formularze do dokonywania wpisów. Analiza plików archiwum nie stwierdziła „dziur” czy niespójności w zapisanych próbkach tekstu.

W sumie program zapisał do archiwum 245 stron (w tym 235 ogłoszeń, reszta to spisy treści i informacje ogólne) o łącznej objętości 250 kB. Lista zindeksowanych stron zajęła kolejne 13 kB, indeks powtarzających się elementów (przydatny tylko jeśli planujemy ponowne archiwizowanie witryny) – 65 kB.

Znacznie gorzej spisało się filtrowanie krótkich elementów. Przy ustaleniu minimalnej długości elementu na 40 znaków, rozmiar archiwum wyniósł ok. 2 MB (799 plików). Wielokrotnie skopiowana została np. przytoczona wyżej informacja o pełnym wyłączeniu odpowiedzialności.

Ta sama zawartość działu „Ogłoszenia” po ściągnięciu programem `wget` zajęła na dysku ponad 4 MB (ok. 800 plików).

`Crawler` (z włączonym filtrem duplikatów i wyłączonym filtrem krótkich elementów) pozwolił zatem na osiągnięcie współczynnika kompresji ok. 15, bez utraty danych istotnych dla badań lingwistycznych. Dla porównania: kompresja ściągniętych stron HTML programem `zip` zmniejszyła ich rozmiar zaledwie trzykrotnie.

Oczywiście zbadany przypadek jest dosyć szczególny, wydaje się jednak na tyle charakterystyczny dla dużych portali internetowych, by warto było stosować opisany wyżej filtr ignorujący powtarzające się elementy.

3.6. Ręczna korekta zawartości archiwum

Przedstawione wyżej metody selekcji automatycznej nie są doskonałe, poza tym zdarza się, że różnego rodzaju błędy popełniają projektanci stron WWW.

W przypadku stwierdzenia, że utworzone archiwum tekstów jest wartościowe, ale znajdują się w nim niewielkie – acz istotne – zaburzenia, można spróbować dokonać ręcznej korekty. Przed przejściem do następnego etapu pracy programu, czyli budowy indeksu kolokacji, możliwe są następujące rodzaje poprawek:

3.6.1. Usunięcie pojedynczych stron

Jeśli jedna lub kilka ze stron zawierają tekst w innym niż reszta próbki języku, zawierają przykład kodu w jakimś języku programowania, *ASCII art* itp., można je usunąć z korpusu

poprzez skasowanie odpowiadającego jej pliku tekstowego w katalogu archiwum. Skasowanie pliku nie utrudnia pracy dalszych modułów, ani nie zaburza poprawności indeksu stron.

3.6.2. Usunięcie fragmentu strony

Z podobnych względów możemy sobie życzyć usunąć z archiwum fragment strony. W tym celu należy skasować odpowiedni fragment odpowiadającego jej pliku tekstowego w katalogu archiwum.

3.6.3. Zmiany w segmentacji tekstu

Również przez edycję odpowiednich plików archiwum można zmodyfikować segmentację tekstu, np. zlikwidować niepotrzebny podział na segmenty lub dodać nowy (usuwanie lub dodawanie znaków końca linii).

3.6.4. Segmentacja na zdania

Aby uniknąć korzystania z niedoskonałego mechanizmu wbudowanego w moduł `IndexBuilder`, można na tym etapie skorzystać z zewnętrznego programu do wykonania już teraz segmentacji na zdania (każde zdanie zapisując w odrębnej linii).

3.6.5. Korekty błędów filtrowania

Najgroźniejszy (choć dość rzadki) błąd filtrowania to wyrzucenie krótkiego elementu (np. pojedynczego podkreślonego lub wytłuszczonego słowa) z dłuższego tekstu. Po stwierdzeniu, że doszło do takiego błędu, można niepotrzebnie wyrzucony element przywrócić poprzez edycję odpowiedniego pliku archiwum.

3.6.6. Korekta błędów źródeł

Zdarza się, że autorzy stron popełniają błędy ortograficzne, interpunkcyjne lub literówki. Z oczywistych względów nie można poprawić źródeł, nic nie stoi na przeszkodzie natomiast, aby dokonać korekty na plikach archiwum. Co więcej, możliwa jest częściowa automatyzacja tej czynności poprzez zastosowanie programu sprawdzającego pisownię, np. `ispell`. Należy jednak zachować przy tym sporą ostrożność, gdyż jedną z istotnych funkcji tworzonego korpusu tekstów może być właśnie wychwycenie nowych słów czy zmian w pisowni.

3.7. Ostateczny efekt działania modułu

W wyniku działania modułu `Crawler` na dysku lokalnym powstaje katalog z plikami tekstowymi (z tekstem dwuwymiarowym) o ujednoczonym kodowaniu (domyślnie UTF-8), reprezentującymi zawartość tekstową zbadanych stron WWW. Nazwy tych plików mają postać `XXXXX.txt`, gdzie `XXXXX` jest liczbą naturalną (w razie potrzeby dopełnioną na początku zerami do pięciu cyfr): np. `00001.txt`, `00002.txt`, `00003.txt` itd.

Dodatkowo w katalogu tym zapisywane są, także w formie plików tekstowych:

- Spis stron, które reprezentują ww. pliki tekstowe

- Lista zignorowanych URL z podaną przyczyną pominięcia, umożliwiającą zdiagnozowanie niezgodnego z oczekiwaniami zakresu pełzania (spowodowanego np. błędem w wyrażeniu regularnym opisującym witrynę lub niemożnością rozpoznania przez program stosowanych przez autora strony odsyłaczy, nieznanymi rozszerzeniami nazw plików).
- W przypadku włączonego filtra duplikatów (rekomendowane): słownik początkowych fragmentów elementów.
- W przypadku zewnętrznego przzerwania działania programu lub wyczerpania ustalonego limitu: kolejka niezbadanych odsyłaczy, umożliwiającą późniejsze wznowienie działania.

Rozdział 4

Kolokacje

Niniejszy rozdział poświęcony jest wykrywaniu kolokacji. Opisuje zaimplementowane metody segmentacji, statystyczne miary siły kolokacji oraz metody filtrowania wyników.

4.1. Kolokacje, ich rodzaje i wykorzystanie

4.1.1. Definicje

Zdania na temat znaczenia terminu kolokacja nie są jednoznaczne. Bez wątpienia kolokacją jest para słów w danym zbiorze tekstów często występujących koło siebie. W zależności od źródła, spotyka się rozszerzenie tego terminu tak, by (a) obejmował również dłuższy ciąg słów często występujących koło siebie lub (b) parę słów występujących w pobliżu siebie (czyli przedzielone nie więcej niż ustalona liczba innych słów).

Bigram – to kolokacja złożona z dwóch słów.

Trigram – kolokacja złożona z trzech słów.

k-gram – kolokacja złożona z k słów.

W pracy niniejszej będziemy rozróżniać trzy typy kolokacji, nazywane (zgodnie z sugestią Janusza S. Bienia) – incydentalnymi, funkcjonalnymi i idiomatycznymi.

Kolokacja incydentalna oznacza przypadkowe wystąpienia wyrazów obok siebie, niespecjalnie ciekawe z punktu widzenia badań lingwistycznych.

Kolokacja funkcjonalna – wyznacza dziedzinę zastosowań danego słowa, ale nie niesie ze sobą znaczenia niemożliwego do wywnioskowania ze słów składowych. Do kolokacji funkcjonalnych można zaliczyć np. *novel character*, *resolve conflicts*, *miasto stołeczne* czy *stolicę Polski*.

Kolokacja idiomatyczna to para lub ciąg słów, którego (pełne) znaczenie lub konotacje nie mogą być wyprowadzone ze znaczenia lub konotacji słów składających się na ciąg. Inaczej rzecz ujmując, istnieje taki poziom semantyczny języka, na którym kolokacja idiomatyczna jest jednostką niepodzielną.

Inne spotykane w literaturze określenia kolokacji idiomatycznej to: jednostka wielowyrazowa (ang. *Multiword Unit*, *MWU* [ScJu][MeAn][ThFK]), definiowalne połączenie wyrazowe [Wier], kolokacja związana (ang. *connected collocation* [ScJu]).

Kolokacją idiomatyczną zarówno w języku polskim jak i angielskim jest np. *compact disc* (*plyta kompaktowa*) – ze znaczenia słów *disc* oraz *compact* nie da się wywnioskować, do czego może służyć *compact disk*.

Kolokacje idiomatyczne często są również niepodstawialne (ang. *un-substitutable*) i/lub niemodyfikowalne (ang. *un-modifiable*).

Wyraz	Popularne kolokacje wyrazu
słowo	kultura, żywe, znaczenie, polskie, użycie
wyraz	szacunku, obcy, znaczenie, słownik, użycie

Tabela 4.1: Porównanie kolokacji wyrazów „słowo” i „wyraz” (wyniki dla zapytań o kolokacje na witrynie Rady Języka Polskiego, odpowiednio dla „słowo słowem słowa słowami słów” oraz „wyraz*”, z pominięciem kolokacji z formami wyrazu „certyfikat”, specyficznymi dla konkretnego artykułu „Stanowisko Rady wobec użycia słowa certyfikat...”)

Niepodstawialność oznacza, że ciąg słów powstały po zastąpieniu jednego ze składników MWU synonimem tego składnika ma inne znaczenie niż oryginalny MWU lub wręcz jest bezsensowną zbitką wyrazów. Nie można np. zamiast *compact disk* napisać *densely-packed disk*, a zamiast *telefon komórkowy* – *telefon zawierający komórki*.

Niepodstawialność drugiego rodzaju – nie synonimy, ale słowa „podobne” – np. *francuski piesek* vs *niemiecki piesek*.

Niemodyfikowalność oznacza, że istotne są nie tylko słowa tworzące MWU, ale również struktura MWU. Nie można np. zamiast *compact disk* napisać *disk that is compact* (w języku polskim niemodyfikowalność jest właściwością rzadziej spotykaną ze względu na swobodniejszy szyk wyrazów).

4.1.2. Wykorzystanie kolokacji

Kolokacje pozwalają zidentyfikować pewne nieuświadomiane na ogół związki i założenia.

Zbiór wszystkich kolokacji występujących w tekście może posłużyć do znalezienia idiomów, zwrotów wymagających objaśnienia, zdefiniowania, umieszczenia w indeksie czy – w przypadku tłumaczeń – przetłumaczenia w sposób spójny, tak by *compact disc* zawsze był *plytą kompaktową*, a nie czasem *plytą*, czasem *dyskiem kompaktowym*, a czasem *zwartym dyskiem*.

Z kolei zbiór kolokacji konkretnego wyrazu pozwala lepiej zrozumieć jego znaczenie, np. rozróżnić zakresy stosowalności synonimów. Z przykładu w tabeli wynika, że o ile w pewnych kontekstach możemy zamiennie używać wyrazów „słowo” i „wyraz”, to istnieją też takie, w których jedno pasuje znacznie bardziej niż drugie – zwroty *kultura wyrazu* albo *słowo szacunku* brzmią przynajmniej sztucznie.

4.2. Automatyzacja wykrywania kolokacji

Przy pracy z korpusami tekstów pożądana jest automatyzacja procesu znajdowania kolokacji. Już przy zbiorach liczących sobie kilka lub kilkadziesiąt tysięcy słów ręczna ekstrakcja kolokacji jest niezwykle żmudna. Tymczasem istnieją korpusy liczące sobie kilka lub kilkadziesiąt milionów wyrazów.

Program, który opisuje niniejsza praca, stosuje do wykrywania kolokacji miary statystyczne (określane też jako probabilistyczne), badając parametry takie jak częstość wystąpienia kolokacji, rozrzut występowania kolokacji pomiędzy teksty wchodzące w skład korpusu lub odchylenie częstości występowania kolokacji od wartości oczekiwanej przy założeniu niezależności słów wchodzących w jej skład. Zaprojektowany został tak, by ułatwić eksperymentowanie z różnymi miarami, umożliwić ich porównywanie i nakładanie na wyniki różnych filtrów.

Warto mieć świadomość, że zaprezentowana tutaj rodzina metod statystycznych nie jest jedynym możliwym podejściem do problematyki wykrywania kolokacji. Piotr Wierchoń w [Wier]

prezentuje np. ekscerpcję bigramów idiomatycznych z tekstu poprzez zastosowanie zestawu prostych wyrażeń regularnych, tworzących filtr, który wykrywa pary słów poprzedzone jednym z określonych ciągów znaków, np. *określa się, określa się mianem, nazywa się, nazwano, tzw.* Ta metoda odznacza się wysoką precyzją (ang. *precision*), ale niską kompletnością (ang. *recall*), to znaczy że wśród otrzymanych za pomocą wyrażeń regularnych wyników jest stosunkowo dużo kolokacji idiomatycznych, ale nie wszystkie kolokacje idiomatyczne znajdują się wśród wyników (w wynikach znajdują się tylko kolokacje jawnie definiowane w tekście).

Dość podobna jest wspomniana w [ScJu] metoda wyszukiwania kolokacji idiomatycznych na podstawie następstwa części mowy – np. *NOUN NOUN, ADJ NOUN* czy *NOUN de NOUN* dla języka francuskiego. Działa ona jednak jedynie na „otagowanym” korpusie tekstów (wyposażonym w znaczniki określające części mowy), dlatego trudno ją stosować do zbiorów tekstów pozyskanych niewielkim nakładem pracy z Internetu.

Warto też zwrócić uwagę, że metody oparte na wyrażeniach regularnych czy częściach mowy muszą być definiowane odrębnie dla każdego języka.

Do zalet metod statystycznych należy:

- uniwersalność – metody statystyczne działają niezależnie od języka i rodzaju tekstów, te same metody można stosować przy budowaniu indeksu dla dokumentacji po angielsku i przy segmentacji tekstu japońskiego na słowa (ale: znajomość specyfiki danego języka pozwala, poprzez zaprojektowanie odpowiednich filtrów, zwiększyć efektywność niektórych metod);
- kompletność – metody statystyczne nie pomijają żadnych istotnych kolokacji, co może się zdarzyć przy ekscerpcji za pomocą wyrażeń regularnych; oczywiście jest to również wadą, bo często powoduje pewną nadmiarowość.

Trzeba też zdawać sobie sprawę z faktu, że nie zawsze interesują nas tylko i wyłącznie kolokacje idiomatyczne. W szczególności dla zastosowań, dla których szukamy kolokacji konkretnego wyrazu, istotny z reguły jest zbiór wszelkich kolokacji, a nie tylko kolokacji znaczących. Dla takich zastosowań wyszukiwanie oparte na wyrażeniach regularnych może okazać się zwyczajnie bezcelowe.

4.3. Segmentacja tekstu i jego podział na słowa

Pierwsze zadanie, z którym musi się zmierzyć moduł wykrywający kolokacje, to podział tekstu (ciągu znaków) na jednostki, które mogą tworzyć kolokacje – słowa.

4.3.1. Podział elementów na segmenty

Podział na segmenty to etap opcjonalny, ale korzystnie wpływający na jakość otrzymywanych wyników.

W pierwszych wersjach programu segmentacja pozyskiwanego tekstu przeprowadzana była jedynie podczas pełzania, na podstawie podziału strony na elementy HTML. Przy próbie wykorzystania pozyskanego tekstu do wykrywania kolokacji okazało się jednak, że taka segmentacja nie wystarcza – stosunkowo często jako kolokacje wskazywane były pary słów rozdzielone np. znakiem otwarcia lub zamknięcia nawiasu.

Dlatego po podzieleniu na elementy i ewentualnym przefiltrowaniu (patrz rozdział 3), przeprowadzany jest drugi poziom segmentacji – na podstawie występujących w tekście nawiasów i znaków interpunkcyjnych. Lista znaków stanowiących granice segmentów definio-

wana jest w pliku konfiguracyjnym. Wystąpienia pary słów rozdzielone jednym ze znaków znajdujących się na liście nie są traktowane jako wystąpienia kolokacji tych słów.

4.3.2. Dualna natura kropki

Problemem nie rozwiązany pozostała interpretacja kropki – stwierdzanie czy dana kropka oznacza w danym kontekście koniec zdania czy skrót. W pierwszym przypadku powinna ona stanowić granicę segmentów, w drugim nie. Być może problem ten zostanie rozwiązany w przyszłych wersjach programu, jest on jednak nietrywialny.

Dla języka angielskiego bardzo często przyjmuje się, że koniec zdania oznaczany jest następującymi po sobie kropką i dwoma spacjami, jednak w istniejących w języku polskim (i zapewne wielu innych) tekstach elektronicznych brak takiego zwyczaju. Trudno się zatem na nim oprzeć, próbując stworzyć w miarę uniwersalny i niezależny od lokalnych zasad program.

Z poczynionych obserwacji wynika, że kolokacje, w których pierwsze słowo jest zakończonym kropką skrótem są umiarkowanie interesujące, np.: *prof. dr, inż. arch., Mr. Jones, jez. Śniardwy, J. Compton*.

Oczywiście, gdyby użytkownik żywił podejrzenia, że traci w ten sposób istotne informacje, możliwe jest usunięcie kropki z listy znaków rozgraniczających.

Kropka występuje też czasem w roli wypunktowania (*I., II.* itp.) lub jako wskazanie, że dany liczbą pełni funkcję liczebnika porządkowego (np. *7.* jako *siódmy*).

4.3.3. Podział segmentów na słowa

Również jeśli chodzi o podział segmentów na słowa, mogą pojawiać się wątpliwości co do znaków stanowiących granicę słów. Dlatego także tutaj zestaw znaków rozgraniczających nie jest zakodowany na stałe, ale opisany w pliku konfiguracyjnym. W szczególności możliwe jest sprawdzenie jak na wyniki wpływa traktowanie lub nie jako granicy słowa znaku -, na stronach internetowych występującego w trzech rolach – myślnika, dywizu i matematycznego minusa. Ciekawym przypadkiem jest też różnica w zapisie niektórych słów w brytyjskim i amerykańskim angielskim (np. *co-ordination* vs *coordination*).

Lista znaków rozgraniczających będzie w pewnym stopniu zależeć od języka. Np. apostrof w języku suahili (a częściowo również angielskim) należy uznać za literę (integralny element słowa), podczas gdy w polskich tekstach apostrof funkcjonuje czasem w roli cudzysłowu, a jako taki nie należy do sąsiadującego z nim słowa.

4.3.4. Czy znaki rozgraniczające są słowami?

Przy tokenizacji (podziale na słowa) tekstu pojawia się pytanie, czy znaków rozgraniczających poszczególne słowa nie należy również uznać za odrębne słowa. W szczególności, w [ScJu] jako słowa traktowane są znaki interpunkcyjne, a jako przykład uzyskanych w ten sposób interesujących kolokacji podano *Dr.* oraz *U.S.* (kropka jest w obu przykładach niezależnym składnikiem kolokacji).

Wydaje się jednak to niepotrzebnym komplikowaniem zadania – wiele kolokacji (np. *commonly-used, web-based, knowledge-free, word-constituent*), które bez znaków rozdzielających składają się z dwóch słów, po uwzględnieniu tych znaków musiałoby mieć trzy składniki. Oznacza to zwiększenie zapotrzebowania na pamięć i przestrzeń dyskową, a także znaczące zwiększenie złożoności obliczeniowej i stopnia komplikacji programu (patrz rozważania na temat niejednoznaczności rozszerzania zakresu stosowania miar statystycznych na dłuższe kolokacje), przy raczej niewielkim zysku jeśli chodzi o otrzymywaną informację. Wydaje się też, że

podanej jako przykład zastosowania w [ScJu] ekscerpcji skrótów można dokonać w prostszy sposób niż angażując mechanizm wykrywania kolokacji.

Innym możliwym pożytkiem z traktowania znaków interpunkcyjnych jako słów jest zmniejszenie szansy na potraktowanie jako kolokacje par (ciągów) słów rozdzielonych takim znakiem. Jednak również w tym przypadku wydaje się, że istnieje prostsze rozwiązanie – w postaci podziału tekstu na segmenty.

Dlatego w obecnej wersji programu znaki rozgraniczające traktowane są jako odstępy. W procedurze tokenizacji pozostawiono wprawdzie możliwość wpływania na sposób traktowania znaków rozgraniczających, ale – bez rozszerzenia procedury znajdowania kolokacji – otrzymywane wyniki będą raczej mało interesujące.

4.3.5. Ignorowanie wybranych słów przy budowie indeksu

Przy próbach zastosowania programu do celów praktycznych, dość szybko okazało się, że – zwłaszcza przy pytaniach o kolokacje konkretnych słów – znaczącą część wyników stanowią kolokacje z krótkimi, często występującymi słowami – przyimkami, spójnikami i innymi *małymi słówkami* (określenie jednej z użytkowniczek). Pierwszym wnioskiem było to, że konieczne będzie rozszerzenie programu, tak by znajdował trigramy.

Lepszym (prostszym) wyjściem okazało się jednak dodanie możliwości ignorowania (traktowania jako odstępy) wybranych słów (przykłady takich słów w tabeli) podczas budowy indeksu.

Dzięki np. specjalnemu potraktowaniu słowa *the*, w obu zdaniach:

- (a) *Smerge mode provides commands to resolve conflicts by selecting specific changes.*
- (b) *Then you can resolve the conflicts by editing the file manually.*

zaobserwowane i odnotowane w indeksie zostanie sąsiedztwo słów *resolve conflicts* (zamiast *resolve the* oraz *the conflicts* w zdaniu (b)), co pozwoli potem następnym modułom programu wysnuć wniosek, że istnieje kolokacja *resolve conflicts*.¹

Lista ignorowalnych słów może zostać określona w specjalnym pliku. Można też zadecydować, że ignorowane mają być wszystkie słowa o niewielkiej długości (mniejszej niż ustalona liczba znaków). Zakres listy i minimalna długość słowa to parametry, które zależą od konkretnego zapotrzebowania. W większości przypadków najskuteczniejsze powinno być ich budowanie eksperymentalne – rozpoczęcie od pustej lub niedługiej listy i stopniowe dodawanie słów, które w otrzymanywnych wynikach użytkownik uzna za niepożądane.

Ignorowanie wybranych słów pozwala znacząco poszerzyć zakres stosowalności metod opracowanych z myślą o bigramach. Większość trigramów zawiera jedno z często występujących, łatwo identyfikowalnych słów jako łącznik. Np. na 10 trigramów przedstawionych w zestawieniu w [ScJu], w siedmiu z nich środkowe słowo należy do zdefiniowanych przez nas jako ignorowalne. W wielu wypadkach metoda działa również na 4-gramy, a nawet 5-gramy (np. *Department of the Interior, Wizards of the Coast, Asia and the Pacific, The Cathedral and the Bazaar*).

4.3.6. Ignorowane słowa a definicja kolokacji

Część użytkowników jako błąd zgłaszało fakt nieuwzględniania *ignorowanych słów* w zestawieniu kolokacji, np. to, że zamiast *noclegów i wyżywienia* w zestawieniu pojawia się *noclegów*

¹Przykład pochodzi z dokumentacji Emacsa, w której *resolve conflicts* występuje na 13 miejscu wygenerowanego rankingu kolokacji według miar Maximum Mutual Information Ratio oraz Pointwise Mutual Information, na 19-21 miejscu według kilku innych.

Język	Przykładowy zestaw słów ignorowalnych
polski	i, o, się, w, we, z, ze
angielski	a, an, and, of, the
suahili	wa, ya, la, cha, vya, za, kwa, pa, mwa, huu, huyu, huo, hawa, hao, haya, hii, hiki, hivi, hizi, hiyo, hicho, hivyo, hizo, hilo, hili

Tabela 4.2: Przykładowe zestawy słów ignorowalnych dla języka polskiego, angielskiego i suahili. Zestaw dla języka suahili został zaproponowany przez Beatę Wójtowicz.

wyżywienia. Pretensja wydaje się słuszna, jednak należy zwrócić uwagę, że w korpusie tekstów para słów (*noclegów, wyżywienia*) może występować w więcej niż jednym trigramie – oprócz *noclegów i wyżywienia* może się pojawić np.:

- *noclegów bez wyżywienia*
- *noclegów lub wyżywienia*
- *noclegów oraz wyżywienia*

Nietrudno sobie wyobrazić zbiór tekstów, w którym występują 4 wyżej wymienione trigramy, oraz filtr ignorujący wszystkie ze słów: *bez, i, lub, oraz*. Który zatem trigram powinien pojawić się w zestawieniu kolokacji jako reprezentacja pary (*noclegów, wyżywienia*)?

Nasuujące się następne pytanie – czy uprawnione jest traktowanie *noclegów i wyżywienia* oraz *noclegów bez wyżywienia* jako jednej kolokacji – jest w istocie pytaniem o definicję kolokacji. Jak zostało wspomniane na wstępie, różne źródła podają różne definicje – jeśli przyjmiemy, że kolokacją jest ciąg bezpośrednio sąsiadujących ze sobą wyrazów, każdy z ww. trigramów powinien zostać uznany za oddzielną kolokację. W tej pracy preferowana jest jednak pojmowanie kolokacji jako związku między dwoma jednostkami – chociażby ze względu na problemy z jednoznacznym definiowaniem i porównywaniem niektórych miar statystycznych w przypadku więcej niż dwóch jednostek.

Warto zwrócić uwagę, że analogiczna sytuacja dotyczy przytoczonego wcześniej przypadku *resolve conflicts*. Czy w zestawieniu powinno się pojawić *resolve conflicts* czy *resolve the conflicts*? Bo przecież traktowanie tych dwóch związków jako odrębnych kolokacji wydaje się bezcelowe.

Za podejściem *kolokacja = para słów* przemawiają też względy praktyczne. Pamiętanie dla każdej kolokacji informacji o ciągu (może być więcej niż jedno) zignorowanych słów przedzielających zwiększyłoby wymagania pamięciowe oraz ilość miejsca zajmowaną przez indeks na dysku o 15-30% (w zależności od średniej częstości występowania kolokacji). Wydaje się, że prezentowana w ten sposób dodatkowa informacja (tak czy owak możliwa do uzyskania poprzez zapytanie o konteksty) nie jest warta obniżenia efektywności programu, przynajmniej dopóki (jeśli) nie zostanie opracowana bardziej efektywna metoda reprezentacji.

4.4. Struktura indeksu

Dla każdego występującego w tekście wyrazu przechowywana jest lista (a dokładniej: tablica o rozmiarze ustalonym w momencie zapisywania indeksu na dysku) tworzonych przez ten wyraz kolokacji. Dla każdej kolokacji – wektor wystąpień w postaci par (identyfikator pliku, pozycja) oraz flagi.

4.4.1. Wystąpienia

Każde wystąpienie przechowywane jest w postaci pary (identyfikator pliku, pozycja). Na takie wystąpienie zarezerwowane zostały 32 bity (4 bajty), z tego 15 bitów na identyfikator pliku, 15 bitów na pozycję w pliku i 2 bity rezerwy. Wydaje się, że to niewiele, jednak dotychczasowe próby zastosowania programu wykazały, że nie stanowi to istotnego ograniczenia możliwości. Na wszelki wypadek w module Crawler dodana została jednak procedura podziału plików wejściowych zawierających więcej niż 2^{15} znaków na mniejsze, możliwe do opisania w zaprojektowanym modelu.

Po modyfikacji ww. procedury, tak by kilka krótszych stron WWW mogło być zapisywanych w jednym pliku lokalnym, program będzie mógł teoretycznie przetwarzać zbiory tekstów o rozmiarach sięgających 2^{30} znaków. Dla języka angielskiego oznacza to ok. 150-200 mln słów. Największe z obecnie istniejących korpusów przekraczają już te rozmiary (np. *Bank of English* – ok. 450 mln słów [Coll]), ale w praktyce korzysta się z znacznie mniejszych próbek korpusu, o wielkościach rzędu od kilkuset tysięcy do kilku milionów słów. W wyżej wymienionym *Bank of English* wykorzystuje się podzbiory od 9 do 26 mln słów, a internetowy korpus języka polskiego PWN liczy sobie 1,8 mln słów [PWN]. Wydaje się, że dla działania programu istotniejsze jest sprawne i wygodne przetwarzanie mniejszych zbiorów tekstów, samodzielnie pozyskanych za pomocą modułu Crawler, niż czysto teoretyczna możliwość przetwarzania na domowym komputerze korpusów o rekordowych rozmiarach.

4.4.2. Flagi

Obecnie wykorzystywane są dwie flagi, mówiące:

- o kierunku kolokacji;
- o tym, czy dana para słów jest nazwą własną czy nie.

4.4.3. Nazwy własne

Nazwy własne są z reguły silnymi kolokacjami. W pierwszych wersjach programu zajmowały sporą część wygenerowanych list kolokacji. Dla wielu zastosowań przydatna okazała się możliwość nieuwzględnienia nazw własnych w zestawieniach kolokacji.

Dlatego podczas budowy indeksu sprawdzana jest kaszta słów wchodzących w skład kolokacji. Za nazwę własną program uznaje taką kolokację, w której wszystkich wystąpieniach w korpusie oba tworzące w jej skład słowa rozpoczynają się dużą literą.

Oczywiście nie jest to metoda idealna, zdarzają się np. całe akapity tekstu pisane dużymi literami. W języku angielskim spotyka się też rozpoczynanie wszystkich słów wchodzących w skład tytułu (np. tytułu rozdziału) dużą literą. Jednak ewentualne pomyłki dotyczą tylko kolokacji występujących rzadko (w jednym konkretnym kontekście), a zatem o stosunkowo niewielkim znaczeniu.

Z drugiej strony zdarzają się jednak nazwy własne, w których jedno ze składowych słów rozpoczyna się z małej litery, np. *Chateau d'Arnise*.

4.5. Przegląd miar statystycznych służących do wykrywania kolokacji

Zaprezentowane tutaj metody statystyczne pochodzą z czterech prac: [Dębo], [ThFK], [ScJu] oraz [YaCh]. Dla porządku warto podkreślić, że prace te same w sobie mają charakter prze-

glądowy i ich autorzy nie są autorami definicji poszczególnych miar.

Oznaczenia:

$c(w)$ – liczba wystąpień słowa w w korpusie

$c(w, w')$ – liczba wystąpień pary słów (w, w') w korpusie

n – całkowita liczba słów w korpusie (rozmiar korpusu)

Zakładamy, że $n \gg 1$.

Rozważane metody statystyczne korzystają z założenia, że prawdopodobieństwo wystąpienia pojedynczego słowa $p(w)$ równe jest częstości jego występowania w badanym korpusie tekstów (ilości wystąpień podzielonej przez rozmiar korpusu) i nie zależy od położenia w korpusie.

$$p(w) = c(w)/n$$

Dla par słów mamy do czynienia z dwoma prawdopodobieństwami:

$$p(w, w') = c(w, w')/n$$

– prawdopodobieństwo obserwowane,

$$P(w, w') = p(w)p(w') = c(w)c(w')/n^2$$

– prawdopodobieństwo oczekiwane przy założeniu, że w i w' są niezależne (dalej określane w skrócie prawdopodobieństwem oczekiwany).

Na podstawie prawdopodobieństwa oczekiwanego $P(w, w')$ wyliczamy oczekiwaną wartość liczby wystąpień pary (w, w') (oczekiwaną przy założeniu, że w i w' są niezależne, dalej określaną w skrócie wartością oczekiwaną):

$$E(w, w') = P(w, w')n = c(w)c(w')/n$$

Większość z niżej wymienionych miar opiera się na różnicy pomiędzy oczekiwany a obserwowany prawdopodobieństwem (ew. oczekiwaną a obserwowaną liczbą wystąpień).

Warto zwrócić uwagę na to, że o ile niektóre miary mają dobrą interpretację teoretyczną (np. odchylenie od wartości oczekiwanej, stosunek prawdopodobieństw, informacja wzajemna), to inne są raczej wynikiem praktycznych eksperymentów (np. formuła Dice'a, LFMD, FSCP).

Wszystkie miary są rosnące, tzn. im wyższa wartość testu dla danej pary słów, tym bardziej prawdopodobne jest, że słowa te tworzą kolokację (lub, wg innej interpretacji: tym silniejszą tworzą kolokację).

Dla niektórych miar (t-score, porównanie prawdopodobieństw) można postarać się wyznaczyć pewną wartość graniczną i rozstrzygnąć, że pary słów o wyższej wartości testu tworzą kolokacje, a pary słów o niższej nie. W praktyce jednak okazało się, że zdecydowana większość par słów choć raz występujących koło siebie spełnia takie wyznaczone metodami teoretycznymi kryterium. Dlatego bardziej od binarnego rozstrzygnięcia czy para słów jest / nie jest kolokacją interesować nas będzie kolejność generowana przez poszczególne miary (dalej określana rankingiem kolokacji).

Po przyjrzeniu się wynikom lub nieco dokładniejszej analizie wzorów, jasne się staje, że wśród miar można wyróżnić dwie odrębne grupy. Do pierwszej – nazwijmy ją częstościową (*frequency-like* [ScJu]) – należą te, które silnie zależą od liczby wystąpień kolokacji. Do drugiej – zależnościowej (*information-like* [ScJu]) – te, które niezależnie od liczby wystąpień osiągają maksimum dla $c(w) = c(w') = c(w, w')$ (w praktyce oznacza to preferowanie kolokacji rzadszych). Do szczególnych przypadków należy zaliczyć PMI, które dodatkowo premiuje kolokacje rzadkie, oraz LFMD i FSCP, które starają się stanowić pewien pomost pomiędzy miarami częstościowymi a zależnościami.

4.5.1. Częstość / liczba wystąpień

Najprostszą miarą tego, czy dane słowa tworzą kolokację, jest ich liczba wystąpień obok siebie w korpusie.

$$R_{freq}(W) = c(W)$$

Taka miara jest jednak zbyt prosta – większość wyników stanowią będą przypadkowe zbitki często występujących słów. Np. w tekstach angielskich do najczęściej występujących par słów należą: *you can, if you, it is, when you, to use, this is, does not, that you, you have, you are* itp.

Dlatego większość miar statystycznych opiera się nie na samej częstości występowania kolokacji, ale na porównaniu częstości występowania pary słów z częstością występowania słów wchodzących w jej skład.

4.5.2. t-score / z-score

Założmy, że słowa w i w' są niezależne. A zatem prawdopodobieństwo że losowo wybraną z korpusu parą następujących po sobie słów będzie (w, w') , powinno wynosić:

$$P(w, w') = p(w)p(w')$$

Wartość oczekiwana liczby wystąpień pary (w, w') w korpusie wynosi przy takim założeniu:

$$E(c(w, w')) = nP(w, w') = \frac{c(w)c(w')}{n}$$

A wariancja:

$$D^2(c(w, w')) = nP(w, w')(1 - P(w, w')) \simeq nP(w, w') = \frac{c(w)c(w')}{n}$$

t-score określa jak bardzo zaobserwowana w korpusie liczba wystąpień pary (w, w') odbiega od wartości oczekiwanej przy założeniu ich niezależności, mierząc różnicę w odchyleniach standardowych.

$$R_z(w, w') = \frac{c(w, w') - E(c(w, w'))}{D(c(w, w'))}$$

$$R_z(w, w') = \frac{c(w, w') - c(w)c(w')/N}{\sqrt{c(w)c(w')/N}}$$

Uwaga: Łukasz Dębowski w [Dębo] zwraca uwagę na to, że statystycy używają nazwy *t-score* na określenie innego testu i proponuje nazwę *z-score*.

Parę (w, w') można uznać za kolokację jeśli różnica jest znacząco większa od odchylenia standardowego. Im *t-score* większy, tym mniejsza szansa, że słowa znalazły się koło siebie przypadkowo. Przy założeniu rozkładu normalnego, za graniczną minimalną wartość można przyjąć *t-score* rzędu 2,5 - 3.

Miara wymieniana w [Dębo] (jako *Test of counts for one count*), [ScJu] oraz [ThFK].

4.5.3. Test chi-kwadrat Pearsona

W poprzednim teście przyjęte zostało założenie, że liczby wystąpień są znacząco niższe od rozmiarów korpusu. Dla bardzo często występujących słów należy wziąć pod uwagę również prawdopodobieństwa ich niewystąpienia.

$$R_{z2}(w, w')^2 = R_z(w, w')^2 + R_z(w, !w')^2 + R_z(!w, w')^2 + R_z(!w, !w')^2$$

Po uproszczeniu ([Dębo]) przybiera postać:

$$R_{\chi^2}(w, w') = (c(ww') - E(ww'))^2 / E(ww')$$

Miara wymieniana w [Dębo] (jako *Test of counts for contingency table*), [ScJu] oraz [ThFK]. W programie zaimplementowana została wersja po uproszczeniach opisanych w [Dębo].

4.5.4. t-score Studenta

T-score Studenta² to odmiana z-score ze zmodyfikowaną wartością wariancji – zamiast oczekiwanej podstawiamy faktyczną liczbę wystąpień.

$$D_{student}^2(c(w, w')) = np(w, w')(1 - p(w, w')) \simeq np(w, w') = c(w, w')$$

$$R_{student}(w, w') = \frac{c(w, w') - E(c(w, w'))}{D_{student}(c(w, w'))}$$

$$R_{student}(w, w') \simeq \frac{c(w, w') - c(w)c(w')}{\sqrt{c(w, w')}}}$$

Miara wymieniana w [ScJu].

4.5.5. Porównanie prawdopodobieństw (LLR)

Miara „porównanie prawdopodobieństw” (ang. *log likelihood ratio*) definiowana jest jako podwojony logarytm ze stosunku prawdopodobieństw wystąpienia pary słów ww' $c(w, w')$ razy przy założeniu hipotezy H1 „słowa w i w' są zależne” do prawdopodobieństwa hipotezy H0 „słowa w i w' występują niezależnie”.

Zgodnie z rozkładem Bernoulli’ego, prawdopodobieństwo, że zdarzenie o prawdopodobieństwie p zajdzie po n próbach dokładnie c razy, wynosi:

$$b(c, n, p) = p^c(1 - p)^{n-c}$$

Dla hipotezy H1 przyjmujemy prawdopodobieństwo pojedynczego zdarzenia:

$$p(w, w') = \frac{c(w, w')}{n}$$

Dla hipotezy H0 przyjmujemy prawdopodobieństwo pojedynczego zdarzenia:

$$P(w, w') = p(w)p(w') = \frac{c(w)c(w')}{n^2}$$

²Student w nazwie testu to pseudonim Williama Gosseta (1876-1937).

A zatem:

$$R_{LLR} = \log \frac{b(c(w, w'), n, c(w, w')/n)}{b(c(w, w'), n, c(w)c(w')/n^2)}$$

$$R_{LLR} = \log((nc(w, w')/(c(w)c(w')))^{c(w, w')}((n - c(w, w'))/(n - c(w)c(w')/n))^{n-c(w, w')})$$

Miara wymieniana w [Dębo], [ScJu] oraz [ThFK].

4.5.6. Poprawione porównanie prawdopodobieństw

Uogólnienie wzoru 4.5.5, tak by nie przybliżał $1 - p(w) = 1$.

Miara wymieniana w [Dębo] jako *log likelihood ratio for contingency table*.

4.5.7. Informacja wzajemna (MI)

Informacja niesiona przez słowo zdefiniowana jest jako minimalną liczbę bitów, na której można zakodować wystąpienie słowa (ciągu słów) w nieskończonym ciągu słów przy stałym rozkładzie prawdopodobieństwa występowania poszczególnych słów.

$$I(w) = -\log_2 p(w)$$

Entropia to [uwaga, uproszczenie $c(w) \ll n$]:

$$H(w) = -p(w)I(w) = -p(w)\log_2 p(w)$$

Informacja wzajemna (ang. *Mutual Information*, MI) [uwaga, uproszczenie jw.]:

$$I(w; w') = H(w) + H(w') - H(w, w')$$

Miara wymieniana w [Dębo].

4.5.8. Przeskalowana informacja wzajemna (MMI)

Informacja wzajemna po przeskalowaniu przez entropię:

$$R_{MMI} = \max\left[\frac{I(w; w')}{H(w)}, \frac{I(w; w')}{H(w')}\right]$$

Miara wymieniana w [Dębo] jako *Maximum Mutual Information Ratio*.

4.5.9. Pointwise Mutual Information (PMI)

$$R_{PMI}(w, w') = I(w) + I(w') - I(ww') = \log_2 \frac{p(w, w')}{p(w)p(w')}$$

PMI można interpretować jako różnicę między liczbą bitów potrzebnych do zapisania (ww') oraz słów (w, w') jako odrębnych jednostek.

PMI bardzo silnie preferuje rzadko występujące kolokacje rzadkich słów. W niefiltrowanych wynikach często pojawiają się kolokacje pochodzące z wtretów obcojęzycznych, będące wynikiem przypadkowych literówek lub rzadkie nazwy własne.

Miara wymieniana w [ScJu] oraz [ThFK].

4.5.10. Symetryczne prawdopodobieństwo warunkowe (SCP)

$$R_{SCP} = \frac{p(w, w')^2}{p(w)p(w')} = \frac{c(w, w')^2}{c(w)c(w')}$$

Iloczyn prawdopodobieństw warunkowych: $\frac{p(w, w')}{p(w)}$ i $\frac{p(w, w')}{p(w')}$. Przyjmuje wartości od 0 (słowa w ogóle nie występują obok siebie) do 1 (słowa występują tylko i wyłącznie obok siebie).
Miara wymieniana w [ScJu] jako *Symmetric Conditional Probability*.

4.5.11. Wzór Dice'a

$$R_{Dice} = \frac{2c(w, w')}{c(w) + c(w')}$$

Przyjmuje wartości od 0 (słowa w ogóle nie występują obok siebie) do 1 (słowa występują tylko i wyłącznie obok siebie).

Miara wymieniana w [ScJu].

4.5.12. Zależność wzajemna (MD)

$$R_{MD} = R_{PMI} - I(ww') = \log_2 \frac{p(w, w')^2}{p(w)p(w')}$$

Miara wymieniana w [ThFK] jako *Mutual Dependency*. Generuje taki sam ranking jak symetryczne prawdopodobieństwo warunkowe, dlatego nie została zaimplementowana.

4.5.13. Log Frequency biased MD (LFMD)

Część miar preferuje kolokacje występujące często (miary częstościowe), część te występujące rzadko (miary zależnościowe). Ciekawe wyniki może dać „skrzyżowanie” miar z obu grup. Intuicja podpowiada, że wśród kolokacji o podobnej wartości zależności, istotniejsze będą te występujące częściej, a wśród kolokacji o podobnej częstości wystąpień, istotniejsze będą te o wyższej zależności.

Według [ThFK], dobre wyniki daje np. dodanie do wartości zależności wzajemnej logarytmu z częstości wystąpień:

$$R_{LFMD} = R_{MD} + \log_2 p(w, w')$$

4.5.14. Frequency biased Symmetric Conditional Probability, FSCP

FSCP to miara zaproponowana przez autora pracy, dająca taką samą kolejność wyników (ranking) jak LFMD, ale nieco prostsza obliczeniowo

$$R_{FSCP} = c(w, w')R_{SCP} = \frac{c(w, w')^3}{c(w)c(w')}$$

4.5.15. Document Frequency

$d(W)$ – liczba dokumentów, w których przynajmniej raz występuje słowo lub ciąg słów W (termin).

Miara sama w sobie nie jest specjalnie przydatna, ale potrzebna do zdefiniowania RIDF oraz niektórych filtrów.

4.5.16. Residual Inverse Document Frequency

Inverse Document Frequency (IDF) może być interpretowane jako liczba bitów, którą zawiera informacja, że w danym dokumencie występuje termin W .

$$IDF(W) = -\log_2 \frac{d(W)}{D}$$

gdzie D – całkowita liczba dokumentów.

Residual Inverse Document Frequency (RIDF) mierzy jak bardzo rozrzut danego terminu pomiędzy dokumentami odbiega od rozrzutu losowego.

$$RIDF(W) = \text{obserwowaneIDF} - \text{oczekiwaneIDF}$$

$$RIDF(W) = -\log_2 \frac{d(W)}{D} + \log_2 [1 - e^{-c(W)/D}]$$

$RIDF$ powinno informować, jak dobrym słowem kluczowym dla osoby szukającej informacji jest W . Jeśli $RIDF(W)$ jest wysokie, to znaczy, że termin W występuje wiele razy w nielicznych dokumentach, a zatem znacząco zawęża zbiór dokumentów. Jeśli $RIDF(W)$ jest niskie, to znaczy, że rozrzut W pomiędzy dokumentami jest mniej więcej równomierny.

4.5.17. RIDF a pozostałe miary

RIDF jest miarą odmienną od pozostałych, do jej obliczenia wykorzystywane są inne cechy badanego ciągu słów. W wynikach różnica objawia się tym, że RIDF lepiej nadaje się do wykrywania terminologii, nazw własnych oraz dobrych słów kluczowych, podczas gdy pozostałe metody lepsze są do zastosowań ogólnych.

Oczywiście, jeśli korpus jest już zawężony do określonej dziedziny, w której interesujące nas terminy zachowują się tak jak terminy popularne, RIDF sprawdza się słabo.

Według [YaCh] dobre wyniki można osiągnąć kombinując RIDF z informacją wzajemną (PMI). Taka kombinacja została zastosowana np. do segmentacji tekstu japońskiego.

Warto zwrócić uwagę na to, że RIDF, w przeciwieństwie do innych miar, można obliczyć zarówno dla ciągu słów (dowolnie długiego) jak i dla pojedynczego słowa – tak naprawdę zatem trudno RIDF określić miarą siły kolokacji.

4.6. Metody filtrowania

Aby zwiększyć stosunek informacji do szumu w wynikach, można próbować odfiltrować kolokacje, które dla danego zastosowania mogą być mało interesujące. Warto jednak zwrócić uwagę, że każda metoda filtrowania (a przynajmniej każda z poniżej przedstawionych) niesie ze sobą ryzyko, że odfiltrowane zostaną również istotne wyniki.

Metody te uzupełniają filtrowanie danych wejściowych, przedstawione w 4.3.5.

4.6.1. Odrzucanie kolokacji rzadkich

Kolokacje występujące tylko raz lub kilka razy w całym korpusie możemy uznać za mało znaczące statystycznie, a wyliczone dla nich miary – za obciążone dużym błędem, a zatem mało wiarygodne. Można znacząco zawęzić zbiór poszukiwań ograniczając się tylko do kolokacji występujących co najmniej T razy (T w zależności od rozmiaru korpusu może być równe np. 2, 5, 10).

4.6.2. Odrzucanie kolokacji pochodzących z jednego źródła

Ta metoda zakłada, że kolokacje występujące w wielu źródłach (w tekstach różnych autorów, na różnych witrynach WWW) są bardziej „wiarygodne”. Jeśli kolokacja pojawia się tylko w jednym określonym tekście, istnieje spore prawdopodobieństwo, że może być efektem specyfiki stylu autora, popełnionej literówki albo rzadką nazwą własną - takie kolokacje są z reguły mało interesujące, zwłaszcza do np. potrzeb konstrukcji słownika.

Oczywiście, można podnieść poprzeczkę wyżej i zajmować się kolokacjami występującymi co najmniej w trzech, czterech lub ogólnie M źródłach.

4.6.3. Odrzucanie kolokacji częstych słów

Niektóre algorytmy („częstościowe”) silnie faworyzują często występujące kolokacje. Niektóre z nich to przypadkowe zbitki często występujących słów. Można zatem spróbować odrzucić kolokacje, które rozpoczynają się lub kończą jednym z N najczęściej występujących słów.

Uwaga: metoda ta w języku angielskim odrzuca np. większość z tzw. *phrasal verbs* (*get about, pick over, throw up* itp.)

4.6.4. Odrzucanie kolokacji zawierających liczby

Kolokacje typu *3 miesiące* czy *rok 2003* są raczej mało interesujące i można z nich zrezygnować.

4.6.5. Odrzucanie nazw własnych

W niektórych metodach (zależnościowych) znaczącą część wyników stanowią nazwy własne. Większość z nich jest mało przydatna np. do potrzeb tworzenia słownika. Dlatego można spróbować ignorować kolokacje, w których we wszystkich wystąpieniach wszystkie słowa rozpoczynają się dużą literą.

4.7. Porównanie efektywności metod

4.7.1. Figure of Merit

Figure of Merit to próba ilościowego porównania efektywności różnych metod statystycznych, zaproponowana w [ScJu], uwzględniająca zarówno *precision*, jak i *recall*.

$$FOM = \left(\frac{1}{H_1} + \frac{2}{H_2} + \dots + \frac{K}{H_K} \right) / K$$

gdzie, dla określonego korpusu, zastosowania i metody tworzenia rankingu:

K – całkowita liczba interesujących kolokacji;

H_i – rozmiar rankingu, który potrzebny jest by zawrzeć i interesujących kolokacji.

Aby wyliczyć FOM, potrzebny jest „obiektywny” zbiór interesujących kolokacji dla danego korpusu (ang. *gold standard*). Zbiór ten wcale nie musi być bardzo duży. Dla potrzeb wyszukiwania kolokacji idiomatycznych mogą to być np. wielowyrzowe hasła w słowniku danego języka.

Dążymy do $FOM = 1$, wtedy pierwszych K kolokacji w rankingu to byłyby tylko i wyłącznie kolokacje interesujące. A jak dobre są istniejące metody?

W [ScJu] dokonano obliczenia FOM przez porównanie wyników rankingów kolokacji otrzymanych dzięki różnym miarom z hasłami zdefiniowanymi w WordNet³. Wartość FOM wahała

³<http://www.cogsci.princeton.edu/~wn/>

się od 0,035 (zwykła częstość), 0,049 (LLR) do 0,167 (formuła Dice'a). Poprzez filtrowanie wyników (odrzućanie nazw własnych, kolokacji występujących tylko w jednym źródle i kolokacji częstych słów) można ten wynik poprawić do 0,152 (częstość) - 0,265 (z-score, chi-kwadrat, SCP).

Podobne rezultaty osiągnięto przyjmując za zbiór interesujących kolokacji połączenia wyrazowe zdefiniowane na witrynach onelook.com, acronymfinder.com i infoplease.com.

Oznacza to, że w najlepszym wypadku w wynikach na jedną interesującą kolokację przypadają trzy nieinteresujące. Zgodne jest to mniej więcej z odczuciami użytkowników programu, jeśli chodzi o wykrywanie kolokacji idiomatycznych. Jeśli do *interesujących* zaliczymy również kolokacje funkcjonalne, osiągamy nieco lepsze wskaźniki efektywności. Wciąż jednak metody statystyczne nie mogą całkowicie zastąpić pracy człowieka (choć mogą ją znacząco wspomóc).

Rozdział 5

Możliwości rozbudowy programu

Niniejszy rozdział przeznaczony jest dla osób zainteresowanych rozwojem programu oraz zaawansowanych użytkowników pragnących na własną rękę dokonać modyfikacji działania programu wykraczających poza zakres objęty plikami konfiguracyjnymi.

Do wykonania tego typu modyfikacji przydatna będzie podstawowa znajomość języka programowania Java.

5.1. Moduł Crawler

5.1.1. Rozpoznawanie języka

Za rozpoznawanie języka odpowiedzialna jest klasa `LanguageFilter`. Można spróbować zmodyfikować w niej metodę `test(content)`, tak by zamiast polegać na znacznikach HTML brała pod uwagę np. występowanie w zawartości tekstowej strony charakterystycznych dla danego języka słowa.

Jeśli powstaną lub zostaną zidentyfikowane witryny internetowe, w konsekwentny sposób stosujące w dokumentach atrybuty `lang` i / lub `xml:lang`, warto byłoby zmodyfikować metodę `Crawler.parse()`, tak by rozpoznawała język nie dla całej strony, lecz odrębnie dla każdego elementu.

5.1.2. Odrzucanie elementów zawierających za dużo błędów

Dla dobrze zdefiniowanego języka (posiadającego dobry słownik) można spróbować podnieść jakość pozyskiwanych próbek tekstu przez filtrowanie na podstawie stopnia zgodności ze słownikiem. Filtr taki odrzucałby elementy, w których mniej niż zadany odsetek słów składowych znajduje się w słowniku. Wybór właściwego poziomu odcięcia może być zadaniem nietrywialnym. Z jednej strony wydaje się, że im wyższy, tym lepszy, a z drugiej – nie może być zbyt wysoki, by (a) w ogóle coś przepuścił i (b) nie wyrzucał tekstów zawierających nazwy własne, słownictwo bardzo specjalistyczne, wyrazy nowe w języku (jeszcze nieobecne w słowniku) lub wyrazy notowane w słowniku w alternatywnej pisowni.

Taki filtr mógłby być przydatny np. przy pobieraniu próbek z archiwów list dyskusyjnych, forów internetowych, ksiąg gości, logów czatów i podobnych współtworzonych przez użytkowników witryn, często wypełnianych bez większej dbałości o zgodność z standardami językowymi. Zadaniem filtra byłoby odróżnianie wiadomości i wpisów napisanych z zachowaniem pewnego minimum poprawności językowej od tych skrajnie niechlujnych. Oczywiście granica jest tu bardzo płynna.

5.1.3. Parser

Zamiast wbudowanego w moduł prostego parsera, można spróbować podłączyć pełniejsze parsery języków HTML (np. wykorzystując pakiet `javax.swing.text.html`) oraz XML (np. SAX) i zamiast na źródle strony operować na drzewie elementów. Niesie to jednak ze sobą ryzyko zwiększenia zapotrzebowania na pamięć dla obecnie stosunkowo mało wymagającego modułu.

5.1.4. Podział plików tekstowych na elementy

Ponieważ program pisany był z myślą o zastosowania sieciowych, podział na elementy opiera się na znacznikach HTML lub XML. Do celów przetwarzania plików lokalnych może się jednak przydać możliwość wstępnego segmentowania plików tekstowych (np. na akapity).

5.2. Indeks kolokacji

5.2.1. Indeksowanie sąsiedztw kolokacji

Można się zastanowić nad modyfikacją struktury indeksu kolokacji, tak by umożliwiała również wyszukiwanie kolokacji na podstawie różnorodności kontekstu (metoda opisana w [MeAn] jako Entropy-based MWU extraction). Rozwiązanie najprostsze – pamiętanie dla każdego wystąpienia sąsiadujących słów – zwiększyłoby rozmiar indeksu dwu-trzykrotnie. Być może sensowniejsza byłaby budowa – z wykorzystaniem fragmentów kodu obecnej klasy `Index` – odrębnego, wyspecjalizowanego indeksu sąsiedztw.

5.3. Miary statystyczne służące do wykrywania kolokacji

Program został tak zaprojektowany, aby ułatwić testowanie skuteczności różnych miar statystycznych. Dlatego zadbano o możliwość łatwego dodawania i modyfikowania różnych testów.

5.3.1. Modyfikowanie istniejących miar

Każda z miar zdefiniowana jest jako klasa w Javie, podklasa klasy `CollocationTest`. Przyjęto, że nazwy klas definiujących miary rozpoczynają się prefiksem `CT` (od *Collocation Test*).

Przykładowa definicja jednej prostej i jednej nieco bardziej skomplikowanej miary:

```
package kolokacje.tests;

public class CTSymmCondProbability extends CollocationTest {

    public String name() {
        return "Symmetric Conditional Probability";
    }

    public String abbrev() {
        return "SCP";
    }

    public float rank(float cww, float cw1, float cw2, float N) {
```

```

        return cww * cww / (cw1 * cw2);
    }
}

package kolokacje.tests;

public class CTCounts41Count extends CollocationTest {

    public String name() {
        return "Test of counts for one count";
    }

    public String abbrev() {
        return "z22";
    }

    public float rank(float cww, float cw1, float cw2, float N) {
        float tmp = cw1 * cw2 / N;
        return (cww - tmp) / (float) Math.sqrt(tmp);
    }

    public boolean test(float rank) {
        return rank > 2.5;
    }

    void setFormat() {
        format.setMinimumFractionDigits(1);
        format.setMaximumFractionDigits(1);
    }
}

```

W definicji występuje od 3 do 5 metod.

abbrev() – zwraca skrót nazwy, używany do identyfikacji miary (np. w pliku konfiguracyjnym) i nazywania plików wynikowych. Zwracana wartość powinna być krótka, jednoznaczna i nie zawierać znaków niedozwolonych w nazwach plików.

name() – pełna nazwa miary, wykorzystywana np. w opisach na generowanych stronach WWW.

rank() – liczbowe ujęcie miary. Zwracana wartość powinna być tym wyższa, im silniejsza według danej miary jest kolokacja danej pary słów (lub: im bardziej prawdopodobne jest, że dana para słów tworzy kolokację). Parametry przekazywane do metody to odpowiednio: liczba wystąpień kolokacji, liczba wystąpień pierwszego słowa, liczba wystąpień drugiego słowa oraz rozmiar korpusu.

test() – metoda opcjonalna, próba binarnego rozstrzygnięcia czy dana wartość miary określa kolokację czy nie. Parametrem jest wynik metody **rank()**, zwracana wartość może być **true** lub **false**. Pary słów, dla których **test(rank())** zwraca **false**, w ogóle nie są uwzględniane w rankingach kolokacji. Jeśli metoda nie zostanie zdefiniowana, wszystkie pary słów traktowane będą jako kolokacje.

setFormat() – metoda opcjonalna, modyfikacja sposobu formatowania. Domyślnie wyniki

przedstawiane są z dokładnością do 3 cyfr po przecinku.

Po dokonaniu poprawek należy skompilować poprawiony plik.

5.3.2. Dodawanie nowych miar

Aby utworzyć nową miarę należy przede wszystkim zdefiniować nową podklasę klasy `CollocationTest` (patrz opis wyżej) i zapisać ją w odrębnym pliku (*nazwaTestu.java*).

Następnie w metodzie `register()` klasy `CollocationTestRegister` należy dodać linijkę:

```
addTest(new nazwaTestu());
```

Ostatnia czynność to skompilowanie zmodyfikowanego pliku `CollocationTestRegister.java`.

5.3.3. Usuwanie miar

Aby usunąć miarę, której obliczanie uznamy za zwykłą stratę czasu, należy w metodzie `register()` klasy `CollocationTestRegister` usunąć odpowiednią linijkę:

```
addTest(new nazwaTestu());
```

a następnie przekompiłować zmodyfikowany plik `CollocationTestRegister.java`.

5.3.4. Miary nietypowe

Program został zoptymalizowany z myślą o miarach statystycznych opierających się na częstości wystąpień kolokacji i jej składników. Jednak możliwe jest też uwzględnienie w zestawieniach miar opierających się na innych danych.

Wymaga to nieco większego nakładu pracy. Oprócz wyżej wymienionych czynności, konieczne jest zdefiniowanie nowego typu miar w klasie `CollocationTest`, uwzględnienie tego typu w klasie `CollocationTestEmptyRegister` oraz utworzenie nowej wersji metody `rank()`, umożliwiającej przekazanie dodatkowych / alternatywnych parametrów.

Przykład implementacji nietypowej miary można znaleźć w pliku `CTResidualInvDocFreq.java`. Miara *Residual Inverse Document Frequency* opiera się na rozrzucie wystąpień danej kolokacji pomiędzy różnymi dokumentami (stara się określić na ile ten rozrzut odbiega od przypadkowego). Wprawdzie kod jest nieco mniej elegancki i czytelny niż w przypadku miar typowych, tym niemniej widać, że wprowadzanie miar korzystających z innego zestawu parametrów jest możliwe.

5.4. Filtrowanie danych wejściowych i wyników

5.4.1. Odrzucanie kolokacji podstawialnych

Pary słów *stolica Polski*, *stolica Litwy*, *stolica Holandii* itp. są kolokacjami, ale stosunkowo mało interesującymi – ich znaczenie wywodzi się wprost ze znaczenia poszczególnych słów. Podobnie np. *pleaded guilty* i *pleaded innocent*. Wydaje się zatem, że dla niektórych zastosowań wyeliminowanie kolokacji funkcjonalnych, w których wymiana składnika modyfikuje znaczenie, ale nie pozbawia sensu, byłoby korzystne dla jakości wyników.

Należy zatem poszukać kolokacji, które różnią się na jednej pozycji. Jeśli na tej pozycji występują słowa różne, ale jakoś związane semantycznie (może *Latent Semantic Analysis* [ScJu], ale może też coś prostszego), to usuwamy taką kolokację z rankingu.

Problem: można wylać dziecko z kąpielą, np. *Al Gore – Albert Gore, bachelor's degree – master's degree*. Sytuacje wątpliwe: *tlenek węgla – dwutlenek węgla*.

Archiwum	Język	Liczba słów	Liczba różnych	Średnia l. wystąpień	Max. l. wystąpień
Dokumentacja Emacsa	angielski	192 tys.	7 800	25 (0,013%)	6009 (3,1%)
Witryna RJP	polski	44 tys.	13 300	3,3 (0,007%)	742 (1,7%)

Tabela 5.1: Porównanie liczby słów i liczby różnych słów dla próbek języka angielskiego i polskiego

Zamiast całkowicie usuwać z rankingu, możemy takie „podobne” kolokacje potraktować jako jedną („skleić”). Wartość miary tej nowej kolokacji może być średnią, medianą, maksimum lub inną kombinacją wartości miar poszczególnych kolokacji. Może też być obliczona od nowa z nową wartością liczby wystąpień.

Według [ScJu] efekty zastosowania takiego filtru są pozytywne, ale bardzo nieznacznie. Osiągnięto poprawę jakości rankingu (mierzonej za pomocą *Figure of Merit* – patrz 4.7.1) rzędu zaledwie 1%.

5.4.2. Sprowadzanie do formy podstawowej

W niektórych językach – np. polskim – jest bardzo duży alfabet słów. Każde słowo występuje stosunkowo niewiele razy, a większość słów występuje w zbiorze tekstów zaledwie jeden lub dwa razy.

Być może dane byłyby łatwiejsze do obróbki statystycznej i wnioskowania o kolokacjach, gdyby w trakcie budowania indeksu sprowadzono wyrazy do formy podstawowej:

poduszek powietrznych – *poduszka powietrzna*

Poradnika Domowego – *Poradnik Domowy*

Pozwoliłoby to znacząco zredukować alfabet słów (do rozmiarów porównywalnych np. z alfabetem dla języka angielskiego).

Możliwe problemy:

1. Czy nie będzie „sklejane” za dużo?
2. Niejednoznaczności w analizie morfologicznej.
3. Forma podstawowa wyrazu może różnić się od słowa tworzącego formę podstawową związku wyrazowego, np. *wyraz szacunku* vs *szacunek*.

Na potrzeby ewentualnego późniejszego podłączenia zewnętrznego analizatora morfologicznego wydzielono procedurę sprowadzania do formy podstawowej do odrębnej klasy – *WordBaseForm*. Obecnie procedura ta jedynie ujednolica kasztę indeksowanych słów.

5.5. Kolokacje dłuższe niż dwa słowa

Ze względu na problemy ze zadowalającym (jednoznacznym i dającym zgodne z intuicją wyniki) zdefiniowaniem miar dla dłuższych kolokacji, zakres działania obecnej wersji programu ograniczony został do kolokacji dwóch słów, ewentualnie rozdzielonych dowolną ilością słów ignorowalnych.

Nie jest to silne ograniczenie – zdecydowana większość kolokacji obejmuje dokładnie dwa słowa. W zestawieniu wyników podanym w [ScJu], gdzie badane były kolokacje o długości do 10 słów, na 99 pozycji zaledwie trzy zajmują kolokacje nie posiadające oczywistej reprezentacji w programie. Co więcej, z tych trzech kolokacji, dwie – *House of Representatives Wednesday* oraz *Sault Ste. Marie* – trudno zaliczyć do specjalnie interesujących (trzecią jest *Hubble Space Telescope*).

Gdyby ktos zdecydował się na rozszerzenie funkcjonalności programu w tym zakresie, musi zmierzyć się z dwoma problemami – metody wyboru i liczenia liczby wystąpień podciągów, oraz przedefiniowaniem miar statystycznych, tak by obejmowały również przypadki kolokacji dłuższych niż dwa słowa.

5.5.1. Złożoność obliczeń

Jeśli dopuścimy dowolną długość kolokacji, to w korpusie zawierającym n słów będziemy mieli do czynienia z $n(n+1)/2$ możliwymi ciągami słów. Przy liczeniu liczby wystąpień, biorąc pod uwagę również długość ciągów, oznacza to trudną do zaakceptowania złożoność obliczeń rzędu $O(n^3 \log n)$.

Segmentacja ogranicza maksymalną długość ciągu słów, a zatem jest w stanie również znacząco ograniczyć złożoność programu. Wciąż jednak dla większych korpusów bezpośrednie rozszerzenie metody stosowanej dla par słów nie wchodzi w grę.

Dlatego typowe podejście do problemu obejmuje wybranie istotnych bigramów – np. występujących przynajmniej T razy – a następnie próby połączenia ich w dłuższe jednostki.

W [YaCh] stwierdzono, że poprzez zastosowanie tablic suffiksowych, sprytnego sortowania i grupowanie ciągów w klasy, zadanie wyznaczenia liczby wystąpień dla wszystkich możliwych ciągów słów można rozwiązać w czasie $O(n \log n)$

5.5.2. Informacja wzajemna

Informacja wzajemna dla dłuższych ciągów słów (za [YaCh]):

$$R_{MI}(w, W, w') = \log p(w, W, w') - \log p(w, W) - \log p(w'|W)$$

$$R_{MI}(w, W, w') = \log c(w, W, w') + \log c(W) - \log c(w, W) - \log c(W, w')$$

5.5.3. Uogólnianie przez szacowanie

Według [ScJu], siłę kolokacji wielowyrazowej można oszacować przez zastosowanie wzoru jako siłę kolokacji dwóch mniejszych jednostek jedno- lub wielowyrazowych:

$$R(W) = R(w_1 w_2 \dots w_i, w_{i+1} \dots w_n)$$

gdzie: $W = w_1 w_2 \dots w_n$, a i dobrane jest tak, by iloczyn $p(w_1 w_2 \dots w_i) p(w_{i+1} \dots w_n)$ był jak największy.

Interpretujemy tutaj (a właściwie definiujemy) kolokację wielowyrazową jako konkatencję dwóch najbardziej prawdopodobnych podciągów.

Spróbujmy policzyć wartości niektórych miar dla dwóch przykładowych trigramów. Za przykłady posłużą: *radę (324) języka (647) polskiego (420)* w zbiorze tekstów z witryny Rady Języka Polskiego oraz *free (102) software (101) foundation (25)* (liczby w nawiasach oznaczają liczbę wystąpień poszczególnych słów składowych w korpusie).

$$R(\text{radyjezykapolskiego}) = R(\text{rady, jezykapolskiego})$$

(bo $324 * 386 > 104 * 420$)

$$R_{Dice}(rjp) = \frac{2 * 104}{324 + 386} = 0,292$$

$$R_{SCP}(rjp) = \frac{104^2}{324 * 386} = 0,086$$

Dla *Free Software Foundation* warto zwrócić uwagę na raczej niezgodny z intuicją punkt podziału kolokacji:

$$R(\text{freesoftwarefoundation}) = R(\text{free, softwarefoundation})$$

(bo $102 * 23 > 47 * 25$)

$$R_{Dice}(fsf) = \frac{2 * 23}{102 + 23} = 0,368$$

$$R_{SCP}(fsf) = \frac{23^2}{102 * 23} = 0,223$$

Kolokacja	Liczba wystąpień	Wzór Dice'a	SCP
rady języka	104	0,214	0,052
języka polskiego	386	0,724	0,548
rady języka polskiego	104	0,292	0,086
free software	47	0,463	0,214
software foundation	23	0,365	0,210
free software foundation	23	0,368	0,223

Warto zwrócić uwagę na to, że przyjęty punkt podziału służy w istocie minimalizacji otrzymanego wyniku. Tak być jednak musi – w przeciwnym wypadku każdy dłuższy tekst (zdanie, kilka zdań, akapit) uznawalibyśmy za kolokację doskonałą. Nietrudno bowiem znaleźć taki podział dłuższego tekstu T na ciągi słów T_1 i T_2 , że $c(T) = c(T_1) = c(T_2) = 1$, co dla wielu metod (zależnościowych) oznacza maksymalny możliwy wynik testu.

5.6. Obliczanie ilościowych wskaźników jakości testów

Można rozważyć uzupełnienie funkcjonalności programu (być może jako odrębny moduł) poprzez dodanie obliczania wartości *Figure of Merit* (patrz 4.7.1) lub innego ilościowego wskaźnika jakości testu dla zadanego (w pliku lub na odrębnym serwerze, np. <http://dict.die.net>) wzorcowego zestawu kolokacji.

5.7. Zastosowanie sieci neuronowej do uczenia programu

Na wielu etapach pracy programu możliwe jest wpływanie na ostateczny wynik poprzez modyfikowanie różnych parametrów. Ponadto, pożądaný wynik – a zatem również wartość tych parametrów – może zależeć od konkretnego zastosowania. Wydaje się, że zamiast wymagać od użytkownika ręcznego doboru parametrów, można by spróbować wykorzystać sieć neuronową do nauczenia programu produkowania oczekiwanych wyników. Dla jakiejś niewielkiej próbki tekstów użytkownik „uczyłby” program poprzez podawanie oczekiwanych wyników, a po osiągnięciu zadowalającej zgodności odpowiedzi programu i użytkownika, program samodzielnie kontynuowałby pracę nad tekstem.

Dwa miejsca, w których takie uczenie mogłoby przynieść szczególnie ciekawe rezultaty, to;

- Selekcja elementów strony na podstawie parametrów takich jak: długość, rozkład liter, występowanie charakterystycznych słów, zgodność ze słownikiem...
- Rozróżnianie kolokacji idiomatycznych, funkcjonalnych i incydentalnych na podstawie wyników różnych testów statystycznych i ewentualnie dodatkowych parametrów.

5.8. Internacjonalizacja i lokalizacja programu

Interfejs użytkownika został napisany w języku angielskim, z myślą o jak najszerszym wykorzystaniu programu. Większość z osób zajmujących się lingwistyką obliczeniową posługuje się językiem angielskim. Tym niemniej, warto rozważyć możliwość dokonania w przyszłości internacjonalizacji i lokalizacji programu.

Stosunkowo najprostsza będzie lokalizacja stron generowanych przez moduł `PrettyPrinter` – w tym celu wystarczy przetłumaczyć szablony stron WWW i podać ścieżkę do nowych szablonów w pliku konfiguracyjnym.

Niespecjalnie trudna powinna być też lokalizacja stron generowanych dynamicznie (PHP). Można w tym celu skorzystać z `gettext`, ale ze względu na niewielką objętość tekstu do przetłumaczenia (nagłówki i tytuły tabel oraz formularze zapytań – w sumie kilkanaście linijek i kilkadziesiąt słów), bardziej sensowne wydaje się ręczne poprawienie kodu PHP (wyrzucenie wszystkich napisów wymagających przetłumaczenia do odrębnego pliku oraz dodanie parametru określającego język).

Stosunkowo najtrudniejsze będzie prawdopodobnie przetłumaczenie napisów zawartych bezpośrednio w kodzie odpowiednich modułów w Javie – informacji diagnostycznych, komunikatów o błędach oraz elementów interfejsu modułów `SAManager` i `SAMain`.

Java posiada wbudowany mechanizm umożliwiający internacjonalizację i lokalizację, zawarty w klasie `java.util.ResourceBundle`. Klasa ta jest wspierana przez narzędzia GNU `gettext`, konwersji między formatem PO a `ResourceBundle` można dokonać wywołując program `msgfmt` z opcją `--java`.

Głównym problemem, z którym będzie się musiała zmierzyć osoba dokonująca internacjonalizacji, będzie prawdopodobnie zamiana konkatenacji napisów na odpowiednie wywołanie `MessageFormat.format()`, np. zamiast:

```
"Opening " + filename + "..."
```

w kodzie powinno się znaleźć:

```
MessageFormat.format("Opening {0}...", new Object[] {filename})
```

Ponieważ w Javie konkatenacja jest wbudowanym operatorem, jest używana znacznie częściej niż w programach napisanych w C (nie bez znaczenia jest też fakt, że napisy wykorzystu-

jące konkatencję są zdecydowanie bardziej czytelne niż te wykorzystujące MessageFormat). Dlatego warto rozważyć automatyzację tego etapu konwersji.

Uwaga: dokonując internacjonalizacji należy zwrócić uwagę na napisy identyfikujące zmienne konfiguracyjne. Należy albo w ogóle ich nie tłumaczyć, albo przetłumaczyć również plik `config.ini`.

Rozdział 6

Podsumowanie i wnioski

W ramach pracy stworzone zostało narzędzie umożliwiające:

- Pobieranie zawartości tekstowej ze wskazanych stron internetowych;
- Wyszukiwanie kolokacji w zgromadzonym zbiorze tekstów;
- Testowanie i porównywanie różnych metod wykrywania kolokacji.

Istotną zaletą programu jest jego czytelny podział na moduły. Oddzielone zostało pozyskiwanie tekstu z Internetu od wykrywania w nim kolokacji, a na tym drugim etapie oddzielono samo wykrywanie od prezentowania wyników. Umożliwia to wykorzystywanie poszczególnych części programu w różnych aplikacjach, a także „recykling” mniejszych fragmentów kodu. Przykładem wykorzystania tej modularności programu są alternatywne interfejsy użytkownika przy dostępie do archiwum – generowanie statycznych stron WWW, dostęp przez PHP i serwer zapytań oraz samodzielna aplikacja Javy. Możliwe jest też np. wykorzystanie modułu pozyskiwania tekstu w programie do zestawiania konkordancji albo wykorzystanie modułów do wykrywania kolokacji na próbkach tekstu pozyskanych w inny sposób.

Na etapie pozyskiwania tekstów udało się opracować prosty, skuteczny i niespecjalnie skomplikowany obliczeniowo filtr umożliwiający wybranie fragmentów zawartości tekstowej witryn WWW, tak by tworzyły zbiór próbek tekstu wartościowy z punktu widzenia badań lingwistycznych, istotnie lepszy od korpusów tworzonych poprzez kopiowanie całości zawartości tekstowej poszczególnych stron.

Jeśli zaś chodzi o wykrywanie kolokacji, stworzone zostało narzędzie umożliwiające sprawdzanie skuteczności i porównywanie różnych testów statystycznych służących do wykrywania kolokacji. Przeprowadzone doświadczenia wykazały, że chociaż testy te ułatwiają proces ekstrakcji kolokacji z korpusu tekstów, daleko im jednak jeszcze do całkowitego wyeliminowania z tego procesu pracy człowieka.

Bibliografia

- [Coll] Witryna Collins Cobuild
<http://www.cobuild.collins.co.uk/>
- [Dębo] Łukasz Dębowski, *Statistical Tests for Detection of Collocations*
<http://www.ipipan.waw.pl/~ldebowsk/manuskrypty/colltests.pdf>
- [MeAn] Magnus Merkel, Mikael Andersson, *Knowledge-lite extraction of multi-word units with language filters and entropy thresholds* In Proc. 2000 Conf. User-Oriented Content-Based Text and Image Handling (RIAO'00), pages 737–746, Paris, France, 2000.
- [PWN] Korpus Języka Polskiego Wydawnictwa Naukowego PWN
<http://korpus.pwn.pl/>
- [ScJu] Patrick Schone, Daniel Jurafsky, *Is Knowledge-Free Induction of Multiword Unit Dictionary Headwords a Solved Problem?*
<http://citeseer.nj.nec.com/schone01is.html>
- [ThFK] Aristomenis Thanopoulos, Nikos Fakotakis, George Kokkinakis, *Comparative Evaluation of Collocation Extraction Metrics*, Proceedings of the Third International Conference on Language Resources and Evaluation (LREC-2002), 2002.
- [Wier] Piotr Wierzchoń, *Automatyzacja ekscerpcji definiowanych połączeń wyrazowych. Filtry wyrażen regularnych*, Przestrzenie informacji, Poznań 2002.
- [YaCh] Mikio Yamamoto, Kenneth W. Church, *Using Suffix Arrays to Compute Term Frequency and Document Frequency for All Substrings in a Corpus* In Proc. of ACL Workshop on Very Large Corpora, pages 28–37, Montreal, 1998.

Dodatek A

Program Kolokacje

Na załączonej do pracy płycie kompaktowej znajdują się:

1. Program Kolokacje wraz ze źródłami oraz plikami konfiguracyjnymi (w katalogu `kolokacje`);
2. Dokumentacja techniczna programu w formacie HTML (w katalogu `api`);
3. Instrukcja użytkownika w formatach PS i PDF oraz jej źródła TeX-owe (w katalogu `doc`);
4. Niniejsza praca w formatach PS i PDF oraz jej źródła TeX-owe (w katalogu `doc`);
5. Licencje GPL i GFDL (odpowiednio w katalogach `kolokacje` oraz `doc`);
6. Przykładowe archiwum wraz z indeksem kolokacji utworzone na podstawie angielskiej dokumentacji GNU Emacsa (w katalogu `emacs`).
7. Przykładowe archiwum wraz z indeksem kolokacji utworzone na podstawie polskiej wersji witryny Projektu GNU (w katalogu `gnupl`).
8. Java 2 SDK, Standard Edition 1.4.2 dla Linuksa i Windows (odpowiednio w katalogach `java-l` oraz `java-w`).