



AGH

AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA AUTOMATYKI

Praca dyplomowa inżynierska

*Kontekstowa konsolidacja i agregacja informacji ze stron internetowych
z wykorzystaniem asocjacyjnych struktur AGDS w celu optymalizacji
ich przetwarzania.*

Autor:

Kierunek studiów:

Opiekun pracy:

Szymon Sobański

Automatyka i Robotyka

dr inż. Adrian Horzyk

Kraków, 2014

Oświadczam, świadomy(-a) odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Serdecznie dziękuję ... tu ciąg dalszych podziękowań np. dla promotora, żony, sąsiada itp.

Spis treści

1. Wprowadzenie	7
1.1. Cele pracy	7
1.2. Zawartość pracy	8
2. Pozyskiwanie treści z Internetu	9
2.1. Zasotsowania robotów internetowych	9
2.1.1. Przeszukiwanie uniwersalne	9
2.1.2. Przeszukiwanie skupione	10
2.1.3. Eksploracja struktury sieci	10
2.1.4. Mirroring	10
2.1.5. Analiza stron internetowych	10
2.2. Działanie pająka internetowego	11
2.2.1. Algorytm	11
2.2.2. Uwagi implementacyjne	11
2.3. Ocena zbioru pobranych stron	14
2.3.1. Świeżość i wiek	14
2.3.2. Algorytmy odświeżania zbioru dokumentów	14
2.4. Charakterystyka zmian sieci w czasie	15
2.5. Kwestie etyczne	16
2.5.1. Identyfikacja	16
2.5.2. Robot exclusion protocol	16
2.5.3. Minimalny czas pomiędzy zapytaniami	17
3. Przetwarzanie danych	19
3.1. Wykorzystanie grafu jako struktury danych	19
3.2. Asocjacyjne grafowe struktury danych	20
3.2.1. Definicja	20
3.2.2. Przedstawienie danych w postaci AGDS	22
3.3. Aktywne asocjacyjne grafy neuronowe	25

3.4. Neuroasocjacyjne grafy wiedzy ANAKG	26
4. Projekt	29
4.1. Wymogi i założenia	29
4.1.1. Budowa	29
4.1.2. Podział aplikacji	30
4.1.3. Wejście/wyjście	31
4.2. Opis komponentów 1. - 4.	31
4.2.1. Struktura przechowywanych informacji	31
4.2.2. Przepływ danych	32
4.3. Opis komponentów 5. - 8.	35
4.3.1. Struktura przechowywanych informacji	35
4.3.2. Przepływ danych	35
5. Implementacja	37
5.1. Wymagania instalacyjne	37
5.2. Implementacja komponentów 1. - 4.	38
5.2.1. Konfiguracja	38
5.2.2. Instalacja i uruchomienie	39
5.3. Implementacja komponentów 5. - 8.	40
5.3.1. Konfiguracja	40
5.3.2. Instalacja i uruchomienie	41
5.3.3. Struktura przechowywanych informacji	42
6. Testy Oprogramowania	45
6.1. Testy poprawności tworzenia struktur AGDS	45
6.2. Inne testy i spostrzeżenia.....	45
7. Podsumowanie	51

1. Wprowadzenie

Wraz z rozpowszechnieniem dostępu do Internetu, można zauważyć zwiększenie ilości produkowanych informacji. Jak wynika z przygotowanego przez organizację IDC raportu, między latami 2009, a 2020, przewiduje się 44-krotny wzrost ilości danych dostępnych w sieci [3]. Jednym z najbardziej odczuwalnych skutków wzrostu ilości i dostępności danych, dotyczącym również przedsiębiorstwa, jest tzw. *Information Overload*, czyli nadmiar czynników branych pod uwagę w różnych procesach decyzyjnych. Wpływa to na obniżenie dynamiki, innowacyjności i konkurencyjności na rynku i ma wyraźny negatywny wpływ na gospodarkę [15].

Z drugiej strony, łatwo dostępne i zróżnicowane dane mogą służyć jako podstawa rozwoju, dając przede wszystkim dostęp do ogromnej bazy wiedzy zgromadzonej w sieci, jak i umożliwiając przedsiębiorcom poszerzenie wiadomości o konsumentach, łatwiejsze badanie rynku i dostosowanie oferowanych usług do potrzeb klientów [5, s. 1-26].

Głównym problemem pozostaje kwestia sposobu porządkowania i filtracji dostępnych danych, w celu ich wykorzystania. Klasycznym przykładem aplikacji służącej do tego celu jest *wyszukiwarka internetowa* - witryna dostępna w sieci, która umożliwia przeszukiwanie Internetu pod kątem wprowadzanych przez użytkownika zapytań.

1.1. Cele pracy

Celem poniższej pracy jest zaproponowanie rozwiązania porządkującego dane pochodzące z sieci internetowej i przechowującego je w postaci grafu asocjacyjnego - AGDS [6, s. 112-117].

Poruszone zostają kwestie budowy oprogramowania eksplorującego sieć i pobierającego treść stron, przechowywania tak zdobytych informacji i przetwarzania ich w celu umieszczenia w grafie. Pokróćce omówione zostają fundamenty teoretyczne takiego rozwiązania, założenia projektowe, implementacja i działanie gotowej aplikacji. Ponadto przedstawione są zalety wykorzystywania tego rozwiązania w praktyce.

1.2. Zawartość pracy

W rozdziale 2 przedstawiono podstawowe informacje dotyczące pozyskiwania danych z witryn internetowych. Zilustrowane są podstawowe wymagania stawiane przed oprogramowaniem trawersującym sieć, tzw. *crawlerem*, poruszone są kwestie wydajności, poprawnej implementacji, oraz opisane są względy etyczne, którymi należy się kierować przy korzystaniu z takiego oprogramowania.

Rozdział 3 prezentuje podstawowe informacje o asocjacyjnych grafach AGDS. Wyjaśnia powód zainteresowania takimi strukturami, wskazuje na ich zalety, przedstawia sposób tworzenia struktur i przytacza przykładowe grafy stworzone za pomocą oprogramowania będącego częścią projektu. Stara się również przedstawić konsekwencje sposobów budowy grafu i możliwość uzyskiwania informacji, zawartych w takich strukturach.

Rozdział 4 dotyczy założeń projektowych aplikacji. Przybliżona zostaje architektura, niektóre założenia techniczne i wytyczne implementacyjne.

Następnie, w rozdziale 5, opisane są szczegóły implementacyjne aplikacji realizującej cele pracy. Przytoczone są zalety wybranych technologii, poruszony jest temat instalacji i uruchomienia oprogramowania.

Rozdział 6 przedstawia przeprowadzone testy oprogramowania, przedstawia wynik działania aplikacji zarówno dla kontrolowanych danych testowych, jak i dla danych pobranych z sieci.

2. Pozyskiwanie treści z Internetu

Pozyskiwaniem treści z sieci zajmuje się grupa oprogramowania nazywana sieciowymi pajakami(ang. *web crawlers*, *web spiders*) lub sieciowymi robotami(ang. *web robots*). Danymi wejściowymi jest zazwyczaj grupa stron, a właściwie adresów URL, nazywana *seedem* [1]. Ich zadanie zazwyczaj nie ogranicza się do pobrania jednej strony, ale do trawersowania całej sieci, bądź w celu pozyskania konkretnych informacji, bądź w celu jej eksploracji. Przy czym w związku z ilością danych dostępnych w Internecie zazwyczaj zakłada się pewne filtry i ograniczenia, w celu zwiększenia efektywności robota i zwiększenia szans na odwiedziny najbardziej wartościowych, dla danego zastosowania, stron.

W związku z dynamiczną naturą Internetu, koniecznym jest ciągle odświeżanie posiadanych już informacji, poszukiwanie nowych stron i ew. eliminacja witryn, które już nie funkcjonują. Wpływa to na charakterystykę działania pajaków internetowych i oprogramowania z nimi współpracującego, które musi być w stanie stale przetwarzać dużą ilość informacji w krótkim czasie. W związku z nieprzerwanym przepływem danych wymagana jest również pełna automatyzacja działania i samodzielne podejmowanie decyzji o odwiedzanych w danym czasie witrynach.

Zadanie, które wykonują roboty internetowe jest tylko z pozoru proste. W istocie zawiera w sobie wiele innych, jak utrzymywanie połączeń sieciowych i obsługa częstych błędów, unikanie “pułapek na pajaki”, czy przestrzeganie względów etycznych. Nie dziwi zatem pogląd założycieli firmy Google, którzy w jednym z artykułów stwierdzili, iż robot sieciowy jest najbardziej wyszukany i najwrażliwszy komponentem wyszukiwarki internetowej[13].

2.1. Zasotsowania robotów internetowych

Roboty internetowe można podzielić wg wykonywanego zadania i zastosowań. Wszystkie kategorie charakteryzują się podobnym algorytmem pobierania stron, główne różnice wynikają z wyboru listy adresów do odwiedzenia, jak i ze sposobów priorytetyzacji pobierania informacji z posiadanej kolekcji URLi.

2.1.1. Przeszukiwanie uniwersalne

Roboty **uniwersalne** (ang. *unviersal*) [10, s. 311-315] mają za zadanie przeszukiwanie sieci, w celach eksploracyjnych i zazwyczaj służą wyszukiwarkom internetowym(Google, Bing, DuckDuckGo itd.).

Roboty **przeszukujące wszecz** mają za zadanie zebranie informacji o jak największej ilości dokumentów, reprezentujących zasoby całej dostępnej sieci. Ich nazwa bierze się z analogii zadań przeszukiwania sieci i przeszukiwania grafu. Zazwyczaj stawiany jest też wymóg wysokiej jakości odwiedzanych stron, co może stać w sprzeczności do wymogu szeroko zakrojonej penetracji Internetu [1].

Roboty **przeszukujące wgłęb** [1] starają się odwiedzać strony odpowiadające w określony sposób potrzebom aplikacji. Sposób doboru stron zazwyczaj dotyczy prostych reguł dotyczących podzbioru adresów URL, domen, rozszerzeń plików, czy używanego języka.

2.1.2. Przeszukiwanie skupione

Kolejny rodzaj robotów przeszukujących sieć stanowią roboty **skupione** (ang. *focused*) [14]. Ich zadaniem jest zbieranie informacji o stronach należących do danej, z góry określonej, dziedziny. W tym celu wykorzystuje się często algorytmy uczenia maszynowego i sztucznej inteligencji [10, s. 327-330], które na podstawie zapytania lub znanego podzbioru dokumentów należących do interesującej użytkownika dziedziny, określają przydatność dokumentów pobieranych przez robota. Przeszukiwanie w ten sposób może odbywać się albo trybie *batchowym*, polegającym na pobieraniu i ocenianiu wielu stron dla zadanych z góry kategorii, niezależnych bezpośrednio od użytkownika, albo w trybie *online*, w odpowiedzi na konkretne zapytanie.

2.1.3. Eksploracja struktury sieci

Crawlers używane są również do badania struktury Internetu i sposobu w jaki zmienia się on w czasie. Ten typ zastosowań jest szczególnie wrażliwy na sposób wybierania kolejnych stron i zbiór punktów startowych. Zostało pokazane [11], iż niewłaściwie dobrane adresy stron startowych prowadzą budowania do obrazu sieci niezgodnego ze stanem faktycznym.

2.1.4. Mirroring

Mirroring jest to sporządzanie kopii istniejących stron internetowych, zwykle w celu poprawy dostępności treści danej witryny. Jest to prosta operacja dotycząca ograniczonego zbioru adresów, dlatego też nie wymaga zaawansowanych algorytmów stosowanych w innych robotach.

2.1.5. Analiza stron internetowych

To zastosowanie związane jest z administracją dużymi serwisami. Robot odwiedza zestaw adresów w celu zbadania poprawności działania strony internetowej, wykrywania możliwych usterek lub analizy pod kątem bezpieczeństwa. Strony takie jak Wikipedia używają wyspecjalizowanego oprogramowania w celu automatyzacji wielu zadań, takich jak zbieranie informacji o podstronach, do których nie prowadzi żaden link w serwisie [1].

2.2. Działanie pająka internetowego

Diagram blokowy 2.1 przedstawia sekwencyjny algorytm działania pająka internetowego. Jest on bardzo prosty i nie nadaje się do zastosowań produkcyjnych, jednak ilustruje podstawowe koncepty charakterystyczne dla tego typu zagadnień. Głównym zarzutem wobec tej struktury jest brak współbieżności, każda strona jest pobierana i przetwarzana sekwencyjnie. Jak wynika z raportu Google(<http://analytics.blogspot.in/2012/04/global-site-speed-overview-how-fast-are.html>) mediana czasu ładowania pojedynczej strony internetowej wynosi powyżej 2 s, można się zatem spodziewać, iż będzie to operacja najdłużej trwająca spośród przedstawionych na schemacie.

2.2.1. Algorytm

Przy inicjalizacji robota na wejście podawany jest zestaw adresów URL, stanowiących tzw *seed*. Używane są one do inicjalizacji bierzącej listy unikalnych adresów URL czekających na odwiedzenie (ang. *frontier*). W każdej iteracji crawler zdejmuje pierwszy adres z listy i pobiera stronę znajdującą się pod tym adresem. Następnie następuje ekstrakcja adresów URL znajdujących się na stronie, normalizacja ich i zapis do listy adresów czekających na odwiedzenie. Finalnie, źródło strony zapisywane jest w lokalnym systemie plików i crawler jest gotowy do pobrania kolejnej witryny.

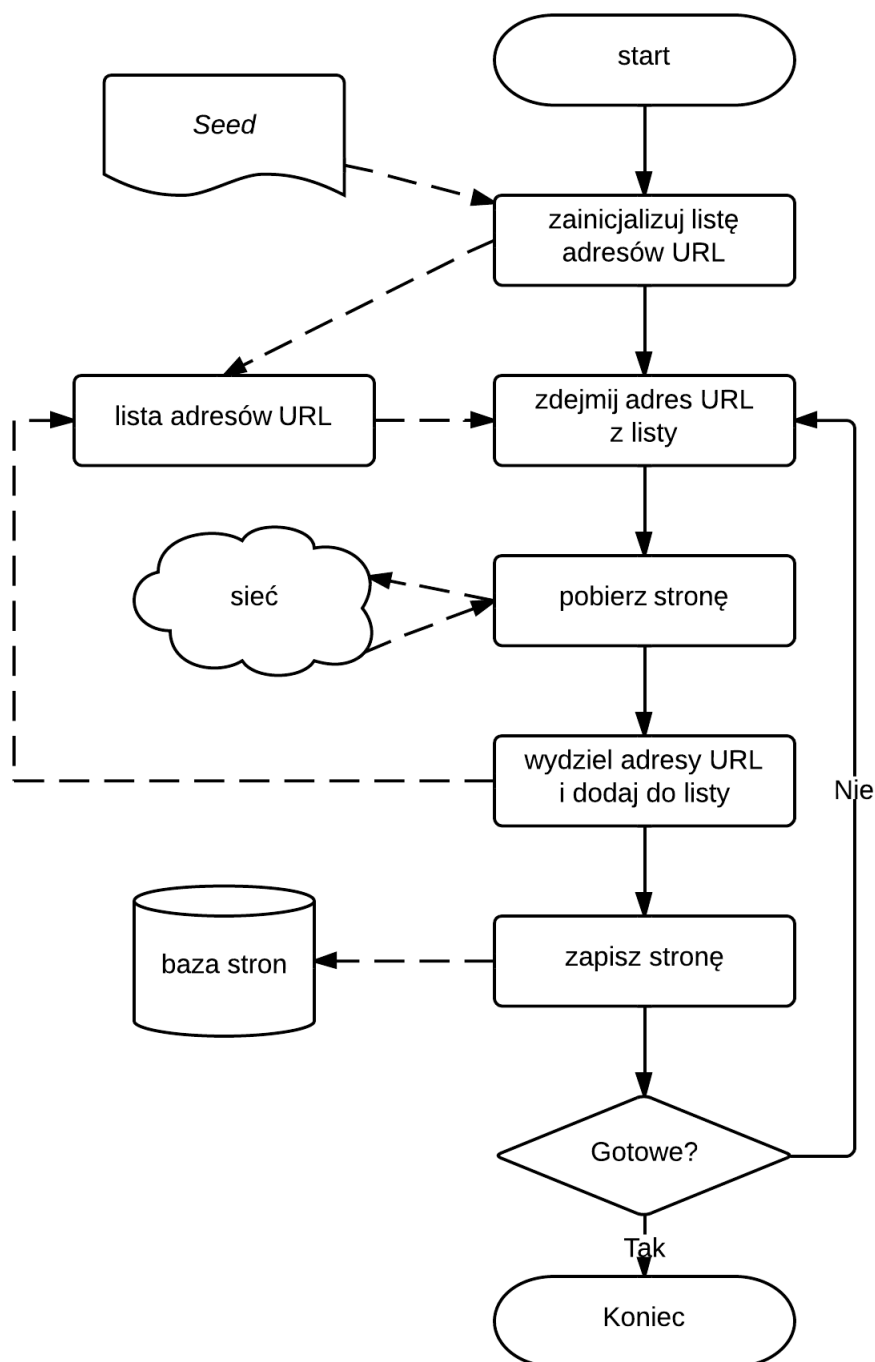
Zazwyczaj algorytm kończy działanie po pobraniu ustalonej z góry ilości stron. Inną możliwością zakończenia działania jest opróżnienie listy oczekujących adresów URL, ale jest to mało prawdopodobne, ze względu na dużą ilość linków - średnio jest to dziesięć adresów URL na stronę[10, s.312].

Dla robotów **przeszukujących wszerez** lista adresów URL implementowana jest zazwyczaj jako kolejka FIFO. Natomiast dla robotów **przeszukujące wgłąb** i **skupionych** używana jest kolejka priorytetowa, z priorytetem przypisanym na podstawie oceny przydatności danego adresu.

2.2.2. Uwagi implementacyjne

W związku ze znaczną ilością scenariuszy możliwych do napotkania przy trawersowaniu sieci, jak i dużą ilością danych, która program przetwarza, stosowane są dodatkowe zabiegi, zwiększające szanse poprawnego działania aplikacji.

1. Stosowanie ścisłej kontroli nad ściąganiem stron: ograniczenie czasu trwania połączenia(stosowanie *timeoutów*), określenie górnej granicy ilości danych pobieranych z jednej strony(np. do 50 KB), wykrywanie nieskończonych pętli przekierowań, logowanie błędów(przekroczenie czasu połączenia, błędne kody odpowiedzi itd.).
2. Preprocessing pobranych źródeł, mający na celu eliminację częstych błędów występujących na stronach, takich jak np. nieodmknięte tagi. Wykrycie kodowania i próba jego normalizacji - np. kodowanie wszystkich pobranych źródeł po stronie crawlera w formacie UTF-8. Na tym etapie



Rysunek 2.1: Schemat blokowy algorytmu robota internetowego.

można przeprowadzić również usuwanie słów nie niosących ze sobą istotnych treści (tzw. *stopwords*), takich jak np. wyrazy “a”, “aby”, “ach”, “aczkolwiek”, ... itd. Stosowane są również inne formy przetwarzania danych, takie jak *lematyzacja* sprowadzenie do podstawowej formy gramatycznej i *stemming* utworzenie tzw. rdzenia słowa, części wspólnej słów niosących podobną

ści sieci, niż w przypadku połączeń synchronicznych. Aby ograniczyć czas spędzony na translacji adresów URL na adresy IP wprowadza się specjalny serwis wykonujący ją zawczasu dla adresów URL oczekujących w kolejce i cache'ujący rezultaty.

2.3. Ocena zbioru pobranych stron

Oprócz efektywnego sposobu pobierania informacji z sieci, ważne jest również zrozumienie cech posiadanego zbioru stron internetowych i jego ewaluacja. W tym przypadku dąży się zarówno do minimalizacji nieaktualnych informacji o witrynach internetowych, jak i do maksymalizacji ilości informacji o stronach w ogóle. Szczególnie ważne jest szybkie reagowanie na zmiany (dodawanie nowych dokumentów, edycję i usuwanie istniejących), które w wypadku sieci postępują nadzwyczaj gwałtownie.

2.3.1. Świeżość i wiek

Ważną cechą stron jest **świeżość** (ang. *freshness*) [10, 1]. Miara ta jest określona dla pobranej strony p w chwili t jako

$$F_p(t) = \begin{cases} 1 & \text{jeżeli } p \text{ jest takie samo, jak lokalna kopia} \\ 0 & \text{w przeciwnym wypadku} \end{cases} \quad (2.1)$$

Dla tej samej pary można określić również **wiek** (ang. *age*) [1], dany zależnością:

$$A_p(t) = \begin{cases} 0 & \text{jeżeli } p \text{ nie jest modyfikowane w chwili } t \\ t - \text{czas modyfikacji } p & \text{w przeciwnym wypadku} \end{cases} \quad (2.2)$$

W zależności od podjętych założeń stosuje się różne algorytmy selekcji dokumentów do ponownego odwiedzenia.

2.3.2. Algorytmy odświeżania zbioru dokumentów

Pierwszy z przedstawionych algorytmów dąży do zmniejszenia liczby stron, które mogą być nieaktualne. W tym modelu nie ma znaczenia historia zmian zapisanej strony, wszystkie dokumenty odświeżane są z taką samą częstotliwością. Wiąże się to z utrzymywaniem dobrej średniej **świeżości**, [2] kosztem dopuszczenia możliwości posiadania nieaktualnych wersji pewnych nadzwyczaj często zmieniających się stron.

Można również stosować inne podejście, które głosi, iż częstotliwość pobierania strony powinna być proporcjonalna do częstotliwości jej zmian w czasie. Wprawdzie takie podejście sprawdza się dla witryn edytowanych regularnie, jednak nadużywane prowadzi do zbyt częstych, nieprzynoszących nowych informacji odwiedzin adresów historycznie zmieniających się z dużą częstotliwością. [2].

Mimo, iż pierwsze podejście jest bliższe optymalnemu, najlepsze rezultaty daje określenie rozkładu prawdopodobieństwa zmian dla poszczególnych witryn i dostosowywanie częstotliwości pobierania informacji z tych źródeł do otrzymanych rezultatów [1].

2.4. Charakterystyka zmian sieci w czasie

Poniżej przytoczona zostaje zbiorcza charakterystyka zmian w różnych podzbiorach sieci, uzyskana z [1].

Tablica 2.1: Zbiorcze dane dotyczące różnych badań zmian zachodzących w sieci.

Podzbiór	Obserwacje
360 losowych stron	mediana życia strony ≈ 2.5 roku, 33% było wciąż dostępnych po 6 latach
500 stron (artykułów naukowych on-line)	mediana życia strony ≈ 4.5 roku
2 500 stron	średnia długość życia ≈ 50 dni mediana wieku ≈ 50 dni
4 200 stron (artykułów naukowych on-line)	mediana życia strony ≈ 4 lat
720 000 stron	średnia długość życia między 60, a 240 dni. 40% stron w domenie .com zmienia się codziennie 50% stron w domenach .gov i .edu pozostaje bez zmian przez 4 miesiące
950 000 stron	średni wiek między 4 dniami, a 4 miesiącami. Strony o dużej ilości linków wskazujących na nie zmieniane częściej.
4 miliony stron	8% tygodniowy przyrost nowych stron 62% stron tygodniowo dodaje nową treść 25% tygodniowy przyrost nowych linków 80% zmian jest niewielka
150 milionów stron	w 10 tygodniowym okresie 65% stron pozostaje bez zmian na 30% stron zachodzą jedynie niewielkie zmiany duża wariacja dostępności, w zależności od domeny
800 milionów stron	średnia długość życia ≈ 140 dni

Jak widać dynamika zmian w sieci zależy w dużym stopniu od indywidualnych witryn, ich miejsca w grafie oraz celu, któremu służą. Praktycznie uniemożliwia to skonstruowanie ogólnego algorytmu odświeżającego posiadaną kolekcję danych w sposób optymalny.

2.5. Kwestie etyczne

Jak wspomniano wcześniej, crawlery używane są w wielu przydatnych zastosowaniach. W związku z tym inwestuje się wiele czasu i pracy w zoptymalizowanie ich działania, od prostych zabiegów pobierania stron asynchronicznie, po zaawansowane techniki modelowania rozkładu prawdopodobieństwa zmian na stronie. Nie należy jednak zapominać, iż działalność profesjonalnych, bardzo wydajnych robotów internetowych nie pozostaje obojętna dla innych użytkowników sieci. Ruch generowany w ten sposób może znacznie obciążać serwery pobieranych stron, powodując kłopoty dla ich administratorów i klientów. Z tego powodu wprowadzono podstawowe zasady, którymi powinien kierować się każdy projektant oprogramowania trawersującego sieć. W momencie pisania tej pracy nie mają one podstaw prawnych, ale łamiąc je notorycznie można liczyć się z potępieniem środowiska i poważniejszymi konsekwencjami, jak np. zablokowaniem puli adresów IP, z których korzysta nieposłuszny robot.

2.5.1. Identyfikacja

Administratorzy stron zazwyczaj zbierają informacje o ruchu, unikalnych użytkownikach, w których dane o działalności robotów nie są zazwyczaj przydatnymi informacjami. Aby ułatwić identyfikację robotów, zaleca się umieszczenie w polu nagłówka zapytania HTTP `user-agent` nazwy crawlera i adresu strony internetowej zawierającej informacje o samym robocie oraz dane kontaktowe organizacji odpowiedzialnej za jego działanie.

2.5.2. Robot exclusion protocol

W celu zapewnienia kontroli nad zasobami, do których dostęp mają roboty internetowe, wprowadzono protokół wykluczający z indeksowania (ang. *robot exclusion protocol*). Przyjmuje on formę statycznej strony internetowej serwowanej przez lokalny dla serwisu adres URL `/robots.txt` [9]. Ma ona podobny układ, jak plik `.txt`, składając się z wielu następujących po sobie linii. Każda linia powinna mieć następujący format: `<pole>:<opcjonalnaspacja><wartosc><opcjonalnaspacja>`. Wg oficjalnej strony [9] pole może być zastąpione przez dwa ciągi znaków:

- `User-agent`: - ciąg znaków odnoszący się do wartości nagłówka `user-agent` w zapytaniu. Podana dalej wartość identyfikuje poszczególne roboty, lub ich grupy. Oznaczanie grup robotów jest możliwe dzięki znakowi szczególnemu `"*"`, który oznacza zero lub więcej wystąpień dowolnego znaku. Po linii zawierającej to pole następuje jedna lub więcej reguł dostępu do witryny.
- `Disallow`: - dyrektywa odnosząca się do stron, których crawler nie powinien odwiedzać. Wartość specyfikuje pojedynczy adres URL, lub ich grupę wyłączoną z indeksacji.

Plik `robots.txt` może zawierać również komentarze, rozpoczynane są one znakiem `"#"`. Wszystkie znaki następujące później są ignorowane. Dodatkowe reguły, respektowane przez niektóre roboty znaleźć

można na ich stronach internetowych. Dla przykładu strona `https://developers.google.com/webmasters/control-crawl-index/docs/robots_txt` przybliża sposób parsowania pliku `robots.txt` przez robota internetowego firmy Google. Ponadto wprowadza ona możliwość grupowania ciągów identyfikujących pole `user-agent` i stosowanie zbioru reguł do całej grupy. Wprowadzona zostaje również dyrektywa `allow` pozwalająca wyszczególnić strony, które robot może odwiedzić.

Poniżej znajdują się przykłady zastosowania przedstawionych wcześniej reguł, zaczerpnięte z [9]:

Listing 2.1:

```
1 # robots.txt for http://www.example.com/
2
3 User-agent: *
4 Disallow: /cyberworld/map/
5 Disallow: /tmp/ # these will soon disappear
6 Disallow: /bar.pdf
```

W tym przypadku żaden robot nie powinien odwiedzać adresu URL zaczynającego się od `/cyberworld/map/`, `/tmp/` lub `/bar.pdf`.

Listing 2.2:

```
1 User-agent: cybermapper
2 Disallow:
3
4 # go away
5 User-agent: *
6 Disallow: /
```

Natomiast ten plik pozwala na dostęp do strony tylko robotowi `cybermapper`.

Ograniczanie dostępu można uzyskać również poprzez umieszczenie w meta tagach HTML klucza `robots` i wartości `noindex` zabraniającej indeksowania danego dokumentu i/lub `nofollow` zabraniającej ściąganie stron linkowanych w dokumencie. Natomiast wartość `nocache` pozwala na pobranie i indeksację dokumentu, jednak nie wyraża zgody na pokazywanie lokalnej kopii użytkownikom usługi wykorzystującej crawler.

2.5.3. Minimalny czas pomiędzy zapytaniami

Aby zniwelować obciążenie związane z działalnością crawlerów wprowadza się wymóg minimalnej ilości czasu, która musi upłynąć między dwoma zapytaniami do tej samej domeny. Spotykane w literaturze propozycje wahają się od 60 do 1 sekundy. Niektóre roboty stosują algorytmy adaptacyjne, uzależniające odstęp między kolejnymi zapytaniami od szybkości odpowiedzi serwisu [1].

3. Przetwarzanie danych

Przedstawione w rozdziale 2 oprogramowanie umożliwia efektywne pobieranie wiadomości z sieci, daje możliwość odświeżania posiadanych informacji i rozszerzania kolekcji dokumentów o nowe pozycje. Rzadko jednak przechowywanie danych jest celem samym w sobie. Oprócz programów archiwizujących, zazwyczaj pobranie dokumentu jest pierwszym etapem algorytmu przetwarzającego dane. Dobrym przykładem jest wyszukiwarka internetowa, w której ważna jest zarówno ilość pobranych stron i ich reprezentatywność na tle całej sieci, jak i algorytm oceny przydatności poszczególnych dokumentów.

Jednym z najważniejszych aspektów tworzenia dobrego silnika przetwarzającego dane jest wybór odpowiedniej, przystosowanej do wykonywanego zadania, struktury danych. Literatura poferuje szeroki wachlarz silników bazodanowych, od relacyjnych, po bazy NoSQL. Niniejsza praca ma na celu przedstawienie wykorzystania struktur grafowych do przechowywania danych ściągniętych z sieci. W tym rozdziale poruszone zostaną pokrótce zalety przedstawienia danych w postaci grafu i omówiony zostanie pewien szczególny rodzaj grafu, jakim jest asocjacyjna grafowa struktura danych (AGDS) [6, s. 108].

3.1. Wykorzystanie grafu jako struktury danych

Struktura grafu wyjątkowo dobrze oddaje bezpośrednie relacje pomiędzy elementami w niej zawartymi, ponadto da się przedstawić w prosty i intuicyjny sposób. Klasycznymi przykładami problemów matematycznych przedstawianych w postaci grafu są np. problem mostów w Królewcu (problem decyzyjny, określający czy w danym grafie istnieje cykl Eulera), czy problem komiwojażera. Wraz z rozwojem internetowych sieci społecznościowych zaczęto zwracać uwagę na możliwości wykorzystania grafów w celu badania sieci, czy przeszukiwania informacji w nich zawartych - przykładem takiego zastosowania jest usługa Graph Search udostępniana przez portal Facebook - <https://www.facebook.com/about/graphsearch>. Poniżej przedstawione zostaną cechy grafu, które decydują o jego rosnącej popularności [7]:

1. **Szybkość** - wyszukanie danych o relacjach pojedynczego wierzchołka nie wymaga kosztownych operacji *join* na całych tabelach, dostęp do danych jest szybki. Nawet w przypadku wzrostu liczby wierzchołków, czas wykonywania większości zapytań nie zmienia się, ponieważ dotyczą one jedynie fragmentu bazy.

2. **Elastyczność** - graf nie wymaga tak dogłębnego określenia struktury bazy, jak tradycyjna baza relacyjna. Dodawanie nowych relacji, węzłów, czy podgrafów jest stosunkowo proste i nie wymaga migracji. Dzięki temu, możliwy jest bardziej dynamiczny rozwój aplikacji opartych o bazy grafowe.

Warto wspomnieć, iż istnieją aplikacje, dla których bazy grafowe są niezastąpione, choćby ze względu na ograniczenia szybkości baz relacyjnych. Przykładem na to jest przytoczone w [8] oprogramowanie szukające relacji typu “znajomy znajomego” w bazie użytkowników sieci społecznościowych. Dla około 1 000 000 użytkowników posiadających średnio po 50 kontaktów, baza grafowa - w tym przypadku Neo4j - okazała się szybsza o kilka rzędów wielkości od tradycyjnej bazy relacyjnej.

Tablica 3.1: Poszukiwanie “znajomych znajomych” w bazie relacyjnej i grafowej

Zagnieżdżenie	Czas wykonania [s]		
	RDBMS	Baza grafowa	Zwrócone rekordy
2	0.16	0.01	≈ 2500
3	30.267	0.168	≈ 110 000
4	1543.505	1.359	≈ 600 000
5	nieukończono	2.132	≈ 800 000

3.2. Asocjacyjne grafowe struktury danych

Często **asocjacja** (skojarzeniem) nazywa się powiązanie ze sobą dwóch lub więcej zdefiniowanych wcześniej elementów. Zgodnie z [6, s. 47] tą definicję można rozszerzyć, opierając się na obserwacji biologicznych organizmów żywych. Zauważono, iż niektóre organizmy mają umiejętność powiązywania dostępnych dla siebie danych w skomplikowane sieci zależności(asocjacji). Do kryteriów, wg których skojarzenia są budowane, należą:

1. Miara podobieństwa.
2. Odległość w czasie recepcji informacji.
3. Kontekst, w którym informacje się pojawiły.

W związku z tym, iż organizmy żywe ciągle pozostają pod wieloma względami niedoścignione w wielu dziedzinach(np. jak analiza obrazu, czy mowy, gdy mowa o człowieku), postanowiono podjąć próbę konstrukcji sztucznych systemów asocjacyjnych, naśladujących te występujące w naturze.

3.2.1. Definicja

Poniższe definicje [6, s. 53-56] opisują podstawowe elementy systemu asocjacyjnego. Na ich potrzebę można przyjąć, iż ma on ogólną budowę grafu mieszanego, w którego węzłach umieszczone są sztuczne neurony, a krawędzie odpowiadają synapsom(połączeniom międzyneuronalnym):

Definicja 1 *Relacją(asocjacyjną) nazywamy każdą relację pomiędzy obiektami(węzłami) zapamiętowanymi i kojarzonymi ze sobą przez system asocjacyjny.*

Definicja 2 *Powiązania łączące podobne i bliskie dane, ich kombinacje i układy nazywane są **powiązaniem asocjacyjnego podobieństwa - ASIM**.*

ASIM ma miejsce wtedy, gdy system przetwarza informacje podobne, jednorodne dane lub ich układy.

Definicja 3 *Powiązania łączące dane, ich powiązania i układy, które następują chronologicznie po sobie lub przestrzennie ze sobą sąsiadują są nazywane **powiązaniem asocjacyjnego następstwa - ASEQ**.*

ASEQ związany jest z powtarzającymi się sekwencjami danych lub danymi pobudzającymi ten sam fragment grafu.

Definicja 4 *Powiązania łączące dane, ich kombinacje i układy w taki sposób, że zwykle samodzielnie nie prowadzą do aktywacji neuronów stanowiących węzły grafu, lecz stanowią tylko pewien kontekst wspomagający ewentualne wywołanie przyszłych aktywnych reakcji neuronów, wywołanych dopiero na skutek następnych procesów skojarzeniowych, nazywa się **powiązaniem asocjacyjnego kontekstu - ACON**.*

ACON to powiązanie niemal analogiczne do ASEQ, z tą różnicą, iż nie skutkuje ono od razu aktywacją neuronu, a pełni tylko funkcję wspomagającą.

Definicja 5 *Bezpośrednie połączenia od receptorów lub neuronów przekazujących dane wejściowe do neuronu, który reprezentuje niektóre kombinacje lub układy zbudowane z tych danych to **powiązania asocjacyjnego definiowania - ADEF**.*

Powstają one w wyniku oddziaływania wielu neuronów, receptorów lub neuronów receptorycznych na ten sam neuron.

Definicja 6 *Grafowa asocjacyjna struktura danych AGDS (ang. associative graph data structure) to graf umożliwiający przechowywanie wartości danych i ich kombinacji wraz z uproszczoną reprezentacją ich asocjacyjnego podobieństwa(ASIM), asocjacyjnego następstwa(ASEQ) i asocjacyjnego definiowania(ADEF), jakie występują między nimi.*

W zapisie formalnym struktura AGDS jest to uporządkowana siódemka $AGDS = (VV, VR, VS, VC, ESIM, ESEQ, EDEF)$ [6, s. 109], gdzie:

VV - zbiór wierzchołków reprezentujących poszczególną wartość (ang. *value vertex*),

VR - zbiór wierzchołków reprezentujących przedział wartości (ang. *range vertex*),

VS - zbiór wierzchołków reprezentujących podzbiór wartości (ang. *subset vertex*),

VC - zbiór wierzchołków reprezentujących kombinację wartości (ang. *combination vertex*),

$ESIM$ - zbiór krawędzi nieskierowanych, łączących asocjacyjnie podobne wierzchołki,

$ESEQ$ - zbiór krawędzi skierowanych łączących asocjacyjnie następne wierzchołki,

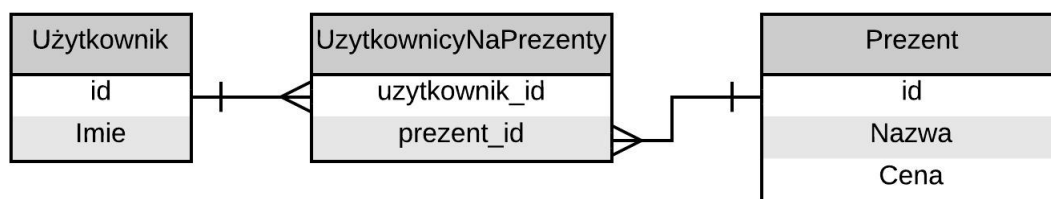
$EDEF$ - zbiór krawędzi dwustronnie skierowanych łączących wierzchołki definiujący z wierzchołkiem definiowanym w taki sposób, iż etykieta (waga) określa przejście od jednego, do drugiego wierzchołka.

Krawędzie dwustronnie skierowane można przedstawić również jako parę krawędzi jednostronnie skierowanych.

Struktura AGDS jest statyczna i z tego względu nie można za jej pomocą modelować innych relacji asocjacyjnych przedstawionych w [6]. Typowo dane odwzorowywane są w niej za pomocą wierzchołków grafu (neuronów), natomiast krawędzie przedstawiają informację o ich wzajemnym powiązaniu, bazującym np. na częstotliwości występowania. Praca z grafem polega bądź na wykorzystaniu istniejących algorytmów przeszukiwania grafu w celu uzyskania interesujących informacji, bądź na przekształceniu struktury AGDS do postaci grafów AANG - **aktywnych asocjacyjnych grafów neuronowych** i stosowaniu algorytmów asocjacyjnych. Ważnym założeniem charakterystycznym dla grafów asocjacyjnych jest unikalność informacji przechowywanej w węzłach.

3.2.2. Przedstawienie danych w postaci AGDS

Na schemacie 3.2.2 przedstawiona jest struktura pewnej relacyjnej bazy danych. Składa się ona z dwóch tabel modelujących relacje: użytkowników i prezentów, jakie otrzymali.



Rysunek 3.1: Schema relacyjnej bazy danych przedstawiająca użytkowników i prezenty, jakie otrzymali

W przypadku używania relacyjnej struktury danych, łatwo zauważyć dwa istotne problemy. Gdy wprowadzona zostaje normalizacja, zazwyczaj informacje nie są duplikowane pomiędzy tabelami, wszystkie zależności wyrażone są w postaci relacji. Jednak operacje na takich zbiorach danych łączą się z koniecznością używania złączeń, które w wypadku tabel o wielu rekordach dają duży narzut obliczeniowy (jak przedstawiono na 3.1). W razie świadomej denormalizacji, jak w przypadku zilustrowanym w tabeli 3.2.2, mogą występować problemy związane z niespójnością danych. Trudno wprowadzać do takiej struktury zmiany, jest ona podatna na różne anomalie.

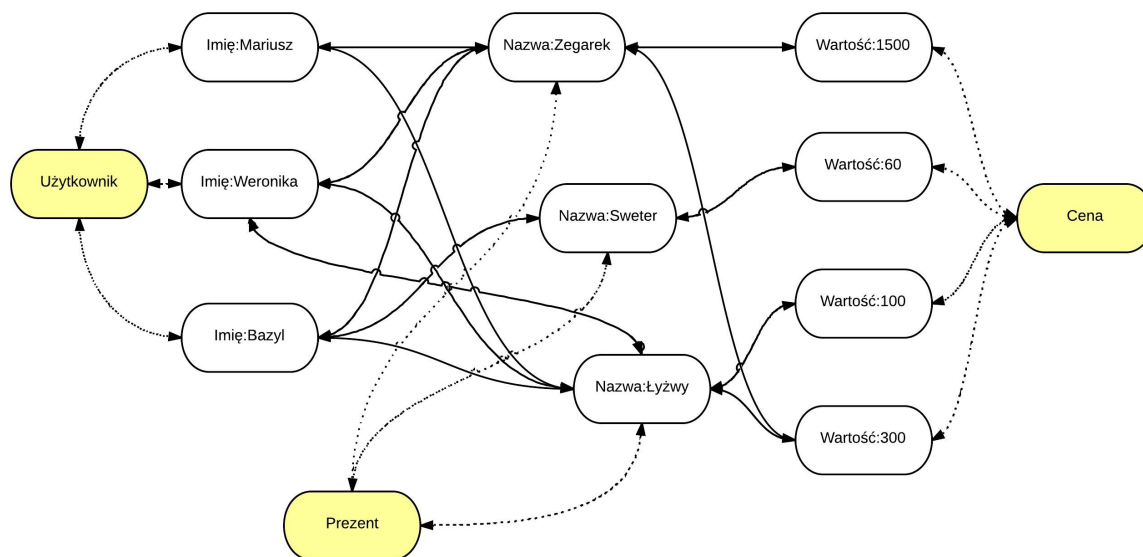
Tablica 3.2: Przykładowe dane

id	Imię	uzytkownik_id	prezent_id	id	Nazwa	Cena
1	Mariusz	1	1	1	Zegarek	1500
2	Weronika	1	2	2	Łyżwy	300
3	Bazyl	2	2	3	Zegarek	300
		2	3	4	Łyżwy	100
		3	4	5	Sweter	60
		3	1			
		3	5			

Tablica 3.3: Tabela joinująca, zaprezentowana w celu zilustrowania danych

Imię	Nazwa Prezentu	Cena
Mariusz	Zegarek	1500
Mariusz	Łyżwy	300
Weronika	Łyżwy	300
Weronika	Zegarek	300
Bazyl	Sweter	60
Bazyl	Zegarek	1500
Bazyl	Łyżwy	100

Na rysunku 3.2.2 przedstawiony jest ten sam zbiór danych, ale z wykorzystaniem struktur AGDS. Niewątpliwą zaletą takiego przedstawienia jest możliwość natychmiastowego otrzymania informacji o relacjach zachodzących między poszczególnymi obiektami. Otrzymanie informacji o liście prezentów użytkownika, wszystkich użytkownikach posiadających ten sam prezent, czy średniej cenie prezentu danego typu jest bardzo proste i niewymagające obliczeniowo.



Rysunek 3.2: Graf reprezentujący dane o użytkownikach i prezentach.

Zgodnie z [6, s. 110] można zauważyć, iż dostęp do każdego obiektu i jego relacji może być zrealizowany w czasie stałym $O(1)$. W przypadku używania relacyjnych baz danych z indeksacją za pomocą B - drzewa - dostępną w MySQLu, czy PostgreSQLu, koszt wyszukiwania wynosi $O(\log n)$. Wprowadzić można stosować index w postaci tablicy z haszowaniem, której średni czas dostępu wynosi $O(1)$, jednak znacznie ogranicza to możliwości operacji na danych posiadających taki klucz.

Rysunek 3.2.2, dla przejrzystości nie uwzględnia etykietowania krawędzi grafu, ani nie przypisuje im żadnych wartości. Jednak zgodnie z definicją 6 nic nie stoi na przeszkodzie, aby wprowadzić różne nazwane relacje, posiadające przypisaną im wartość, lub zbiór wartości. Dzięki temu struktury AGDS mogą być używane do przechowywania skomplikowanych, wielopoziomowych struktur danych. Warto zwrócić uwagę również na łatwość rozszerzania istniejącej struktury o nowe typy węzłów, węzły i łączące je relacje.

Podsumowując, struktury AGDS mogą być użyte wszędzie tam, gdzie tradycyjne struktury grafowe, dając przy tym nowe możliwości związane z modelowaniem relacji asocjacyjnych. Szczególnie można wyróżnić następujące zastosowania [6]:

- jako alternatywna, asocjacyjna forma reprezentacji i przechowywania danych,
- do modelowania podstawowych asocjacyjnych relacji pomiędzy danymi,

- do szybkiego wyszukiwania powiązanych danych i ich kombinacji,
- do szybkiego odnajdywania podobieństw, różnic i korelacji,
- jako pewna forma kompresji danych i ich kombinacji,
- jako pośrednia forma reprezentacji danych między pasywnym klasycznym oraz alternatywnym asocjacyjnym modelem przetwarzania danych.

3.3. Aktywne asocjacyjne grafy neuronowe

Jedną z możliwości wykorzystania struktur AGDS jest przekształcenie ich w asocjacyjne grafy neuronowe - **AANG**. Zazwyczaj AANG nie jest używane *per se*, ale dostosowywane do konkretnych aplikacji, determinując również sposób budowy grafów AGDS.

Założeniem stojącym za grafami AANG jest chęć modelowania biologicznej tkanki mózgowej. Podobnie, jak w przypadku innych sztucznych sieci neuronowych, konieczne jest stosowanie pewnych uproszczeń, umożliwiających sprawne budowanie, uczenie i ewaluację takich struktur. W tym przypadku nacisk położony został głównie na modelowanie odpowiedzi neuronu w czasie. Dlatego obliczanie propagacji sybału przez graf AANG polega na symulowaniu, w czasie ciągłym lub dyskretnym, interakcji neuronów wchodzących w jej skład.

Jako, iż niniejsza praca nie skupia się w znacznym stopniu na symulacji odpowiedzi grafów AANG, wyszczególnione zostaną jedynie najważniejsze cechy neuronów asocjacyjnych. Pełną definicję można znaleźć w [6].

Definicja 7 *Asocjacyjny model neuronu - ASN (ang. associative neuron) jest modelem funkcjonalnym biologicznego neuronu odwzorowującym jego plastyczne zmiany w czasie oraz reaktywne dynamiczne sumowanie ważonych bodźców wejściowych w czasie z uwzględnieniem funkcji relaksacji i refrakcyjności.*

W dalszej części pracy pojęcia *neuron* i *neuron asocjacyjny* są stosowane zamiennie.

Definicja 8 *Aktywne asocjacyjne grafy neuronowe - AANG(ang. active associative neural graphs) to pewien rodzaj sieci neuronowych opartych na strukturach AGDS zdolnych do aktywnego, asocjacyjnego wiązania danych, ich kombinacji i układów oraz wykonywania na nich neuroobliczeń grafowych.*

Grafy AANG posiadają wszystkie zalety i cechy struktur AGDS, przy czym węzły tego grafu zamieniane są w aktywne i warunkowo reaktywne neurony, zaś krawędzie w asocjacyjne połączenia pomiędzy nimi.

3.4. Neuroasocjacyjne grafy wiedzy ANAKG

Jednym z zastosowań grafów AANG opisanych w sekcji 3.3 jest asocjacyjne modelowanie wiedzy. Sprowadza się ono do przetwarzania pewnego zdefiniowanego typu informacji i zapisywania go w postaci grafów AGDS, z możliwością przekształcenia do struktur AANG. Jest to zagadnienie szerokie i nie zostanie dogłębnie zanalizowane. Przytoczona zostanie jedynie część pojęć i algorytmów algorytmów, szczególnie związanych z budowaniem struktury AGDS, potrzebna do zrozumienia działania opisywanego projektu.

Definicja 9 *Wiedza skojarzeniowa jest to wiedza formułowana pod wpływem kombinacji i układów danych oraz ich sekwencji, które można zapisać w postaci zbioru sekwencji uczących \mathbb{S} .*

Zbiór sekwencji uczących jest podstawowym nośnikiem wiedzy skojarzeniowej i może być wykorzystany do uczenia specjalnie do tego przygotowanych grafów AANG.

Definicja 10 *Aktywne asocjacyjne grafy wiedzy - ANAKG (ang. active neuroassociative knowledge graph) są pewnym rodzajem grafów AANG o specyficznej postaci dyskretnej symulacji czasu aktualizowanego w synchronicznie równoległych krokach dla wszystkich jego elementów [6, s. 234].*

Budowa grafu ANAKG zaczyna się od zdefiniowania sekwencji uczącej $S \in \mathbb{S}$ i pustego grafu ANAKG. Następnie rozpoczyna jest symulacja dyskretna z krokiem Δt ,

$$\begin{cases} t_0 = 0 \\ t_i = t_0 + \Delta t * i \\ t_{max} = t_0 + \Delta t * \text{card}(S) \end{cases} \quad (3.1)$$

W chwili t_i podawany jest na wejście ANAKG i -ty element sekwencji uczącej, powodując utworzenie lub aktywację odpowiadającego mu neuronu. Taka aktywacja nazywana jest **aktywacją zewnętrzną**, w przeciwieństwie do **aktywacji wewnętrznej** spowodowanej sygnałami płynącymi od sąsiadujących neuronów. Na podstawie czasu, który upłynął między aktywacjami poszczególnych neuronów można określić tzw. **współczynnik aktywności** połączenia.

Definicja 11 *Współczynnik aktywności połączenia między neuronem presynaptycznym SN aktywowanym w chwili t_{presyn} , a neuronem postsynaptycznym \widehat{SN} aktywowanym w chwili $t_{postsyn}$ definiujemy jako [6, s. 224]:*

$$\begin{cases} \delta_{SN, \widehat{SN}}^{act} = \delta_{SN, \widehat{SN}}^{act} + \frac{1}{\tau} \\ \tau = t_{postsyn} - t_{presyn} \end{cases} \quad (3.2)$$

Ze względu na specyfikę symulacji 3.1 stopień asocjacji ACON jest równy τ . Mimo iż teoretycznie można tworzyć połączenia o dowolnym stopniu asocjacji, tj. traktować całą sekwencję uczącą jako jeden

kontekst, proponuje się jego ograniczenie [6]. Pozwala to na modelowania uogólniania i skończonej pojemności pamięci występującej u organizmów żywych. Dzięki ograniczonemu kontekstowi zmniejsza się również komplikacja grafu i nie tworzone są powiązania pomiędzy niezwiązanymi ze sobą neuronami.

W ramach kontekstu definiowane są połączenie między następującymi po sobie neuronami. Ich wagi mogą być obliczone na podstawie zależności [6, s. 226]:

$$\begin{cases} w_{SN, \widehat{SN}}^{ACON} = \frac{2\delta_{SN, \widehat{SN}}^{act}}{\eta_{SN}^{act} + \delta_{SN, \widehat{SN}}^{act}} \\ \delta_{SN, \widehat{SN}}^{act} = \sum_{(\rightsquigarrow ACON_\tau: SN \rightsquigarrow \dots \rightsquigarrow \widehat{SN} \in AAT)} \frac{1}{\tau} \end{cases} \quad (3.3)$$

gdzie

η_{SN}^{act} - ilość aktywacji neuronu SN dla określonego zbioru sekwencji uczących S .

$\delta_{SN, \widehat{SN}}^{act}$ - współczynnik skuteczności połączenia synaptycznego $SN \rightsquigarrow \widehat{SN}$ (Definicja 11).

Dla zachowania integralności opisu przytoczony zostanie jeszcze wzór pozwalający obliczyć pobudzenie pojedynczego neuronu w chwili t [6, s. 227]:

$$\begin{aligned} exc_t^{SN} &= NRF_{\alpha, \beta}^{SQR}(exc_{t-1}^{SN}, \theta^{SN}, x_1^t, \dots, x_k^t) = \sum_{k=1}^K w_k x_k^t + \\ &+ \begin{cases} \alpha \cdot exc_{t-1}^{SN} + \frac{(\alpha-1) \cdot \beta \cdot (exc_{t-1}^{SN})^2}{\theta^{SN}} & exc_{t-1}^{SN} < 0 \\ \alpha \cdot exc_{t-1}^{SN} + \frac{(1-\alpha) \cdot \beta \cdot (exc_{t-1}^{SN})^2}{\theta^{SN}} & 0 \leq exc_{t-1}^{SN} < \theta^{SN} \\ -\beta \cdot \theta^{SN} & exc_{t-1}^{SN} \geq \theta^{SN} \end{cases} \end{aligned} \quad (3.4)$$

,gdzie

α reluguje szybkość relaksacji neuronu, czyli jego powracanie do stanu spoczynku. Przyjmuje się $0 \leq \alpha < 1$. Wzrost wartości tego współczynnika wpływa na wydłużenie okresu relaksacji.

β odpowiada za czas relaksacji neuronu w stanach podprogowych i poaktywacyjnych. Przyjmuje się $0.5 \leq \beta < 1$.

θ^{SN} to próg aktywacji neuronu. Przyjmuje się $\theta = 1$.

4. Projekt

W tym rozdziale opisana jest realizacja praktyczna projektu. Najpierw omówione zostaną wymagania stawiane całej aplikacji i podstawowe założenia związane z jej implementacją. Następnie przybliżone zostaną jej poszczególne komponenty.

4.1. Wymogi i założenia

4.1.1. Budowa

Podstawowym celem projektu jest implementacja oprogramowania pobierającego z podzbioru stron opublikowanych w Internecie dokumenty i przedstawiającego je w postaci omówionych w Rozdziale 3 asocjacyjnych grafów AGDS. Jest to zadanie wieloetapowe, wymagające następujących komponentów:

1. Roboty dostarczającego dane z sieci. Komponent musi mieć możliwość asynchronicznego pobierania stron WWW, udostępniać API do ekstrakcji adresów URL z ostatnio ściągniętych stron spełniających podane warunki oraz implementować *Robots Exclusion Protocol*.
2. Komponentu współpracującego z wyżej opisanym modułem, zapewniającego interfejs umożliwiający zapisywanie danych do zewnętrznej bazy. Umożliwiającego zapisywanie kolejnych wersji stron WWW, opisanych *timestampem*.
3. Modułu przetwarzającego wybrane strony WWW przechowywane w zewnętrznej bazie. Powinien on być dawać możliwość parsowania strony HTML i operowania na drzewie DOM. Powinien mieć możliwość zapisu informacji w formacie JSON, przy zachowaniu budowy umożliwiającej proste rozszerzanie funkcjonalności poprzez dodawanie odpowiednich klas do projektu.
4. Prostego klienta kolejki RabbitMQ, wykonującego RPC z przetworzonymi wcześniej danymi, jako argumentem.
5. Serwera odbierającego wywołania przez kolejkę RabbitMQ, rozpoczynającego konwersję danych do formatu AGDS i zwracającego wynik na kolejkę.
6. Modułu przetwarzającego dane odebrane przez serwer, dzielącego wysyłane strony na sekwencje uczące, umożliwiającego w razie potrzeby proste rozszerzenie funkcjonalności.

7. Silnika asocjacyjnego współpracującego z warstwą zapewniającą persystencję. Jego jedynym zadaniem jest budowa grafu AGDS zgodnie z założeniami opisanymi w Rozdziale 3.
8. Warstwy pośredniczącej pomiędzy aplikacją, a bazami danych. Powinna ona zapisywać logiczną strukturę grafu w bazie grafowej, a dodatkowe informacje, takie jak atrybuty poszczególnych węzłów (np. częstość występowania), czy krawędzi (np. waga) przechowywać w relacyjnej bazie klucza - wartości. Ważne jest ukrywanie szczegółów implementacyjnych, zwłaszcza wynikających z używania wielu rodzajów baz danych.

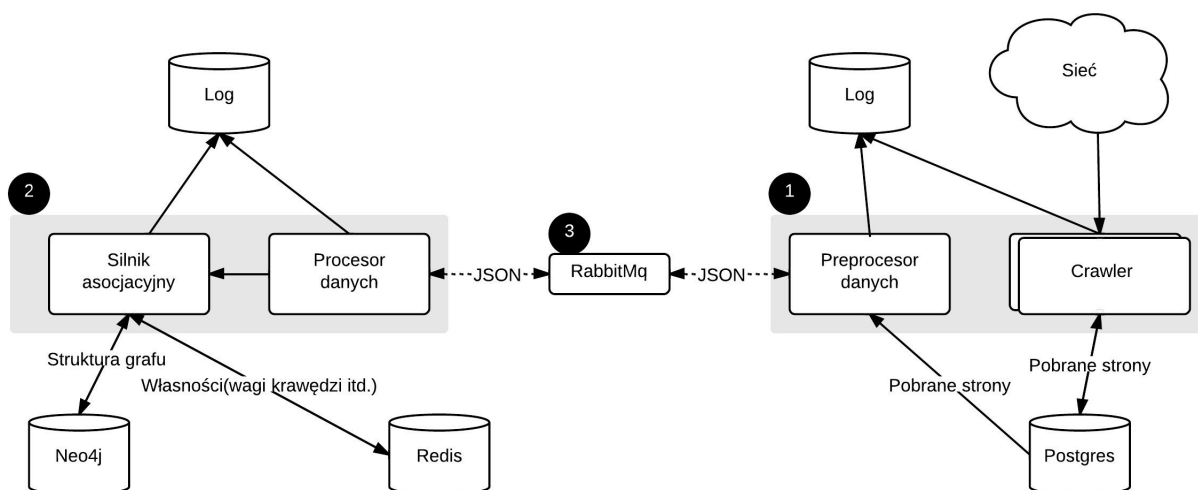
W związku ze skalą projektu i trudnym debugowaniem komunikujących się ze sobą modułów, każdy komponent posiada testy jednostkowe.

4.1.2. Podział aplikacji

W celu lepszej organizacji kodu, jak i separacji odpowiedzialności poszczególnych elementów aplikacji, postanowiono wydzielić z niej dwa podstawowe moduły:

1. Składający się z komponentów 1. - 4. moduł odpowiedzialny za ściąganie i przechowywanie danych z sieci. Może on być traktowany jako repozytorium nieprzetworzonych danych, które można używać wielokrotnie dla różnych eksperymentów.
2. Składający się z komponentów 5. - 8. wyspecjalizowany moduł odpowiedzialny za budowę i przechowywanie struktury AGDS. Jest on niezależny od konkretnego źródła danych, wymaga jedynie ich określonego formatu.

Oba moduły są połączone kolejką z zaimplementowanym protokołem RPC.



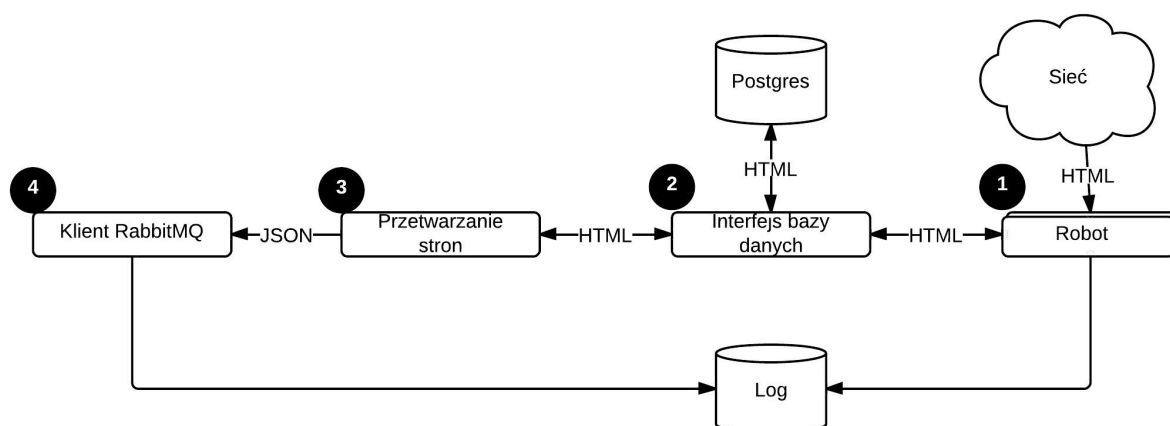
Rysunek 4.1: Schemat architektury aplikacji. 1. - część pobierająca dane, 2. - część konstruująca graf 3. - kolejka

4.1.3. Wejście/wyjście

Aplikacja nie posiada interfejsu graficznego. Poszczególne komponenty są wzbudzone z poziomu *shella*, konfiguracja odbywa się poprzez edycję wydzielonych plików konfiguracyjnych, opcje podawane przy wykonywaniu skryptów lub zmienne środowiskowe. Celem tej części projektu nie jest serwowanie użytkownikowi informacji, a przedstawianie danych w pewny określonym formacie, stąd takie ograniczone spektrum możliwości interakcji z aplikacją. Konkretnie skrypty uruchamiające aplikacje, opcje ich wywoływania i struktura plików konfiguracyjnych opisana jest w dalszej części pracy.

4.2. Opis komponentów 1. - 4.

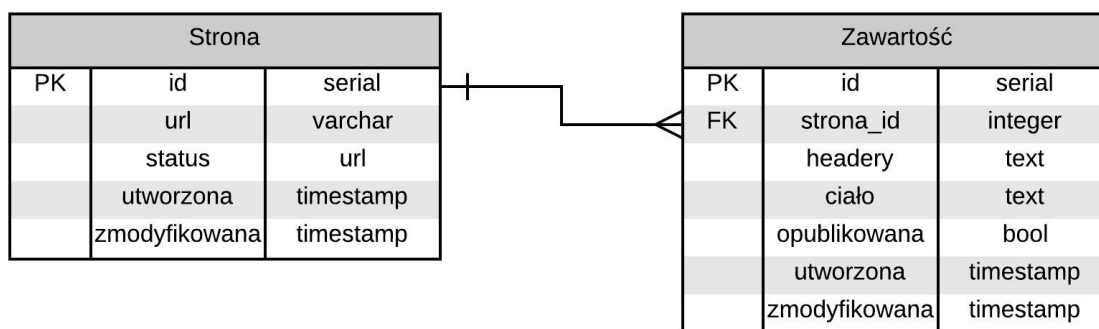
Na rysunku 4.2 przedstawiony jest detaliczny schemat pierwszych czterech komponentów głównej aplikacji. W istocie tworzą one autonomiczną subaplikację, powiązaną z resztą komponentów jedynie poprzez kolejkę..



Rysunek 4.2: Detaliczny schemat komponentów 1. - 4. Podpisy na strzałkach odnoszą się do formatu, w jakim przedstawiane są strony internetowe na każdym z prezentowanych etapów.

4.2.1. Struktura przechowywanych informacji

Struktura bazy umożliwia przechowywanie unikalnych adresów URL w relacji *Strona* i zapisywanie historii kolejnych odwiedzin w osobnej tabeli *Zawartość*. Dzięki temu otrzymuje się prosty sposób na archiwizację nieaktualnych rekordów, przy braku konieczności modyfikacji bazy. Następnie takie rekordy mogłyby być cyklicznie przenoszone np. do hurtowni danych, w celu późniejszej analizy. Dodatkowego wyjaśnienia wymagają dwa pola: pole *strona.status* przyjmujące wartości ze zbioru: {oczekuje, pobierana, sukces, blad}, odpowiadające cyklowi pobierania i przetwarzania informacji. Wartość *pobierana* ma na celu niedopuszczenie do pobierania tej samej strony przez dwa procesy



Rysunek 4.3: Schema przechowująca informacje pobrane z sieci.

robota. Drugie to pole `wartość.opublikowana` odnosi się do faktu wysłania danej zawartości do kolejki. Jeżeli to nastąpiło i komponent nie otrzymał w odpowiedzi zwrotnej komunikatu o błędzie, wartością tego pola wynosić będzie `TRUE`, w przeciwnym razie `FALSE`.

4.2.2. Przepływ danych

Aplikacja posiada dwie niezależne ścieżki przepływu i przetwarzania danych:

1. Ścieżka związana z pobieraniem danych z sieci.
2. Ścieżka przetwarzania ściągniętych danych i publikacja ich za pomocą kolejki.

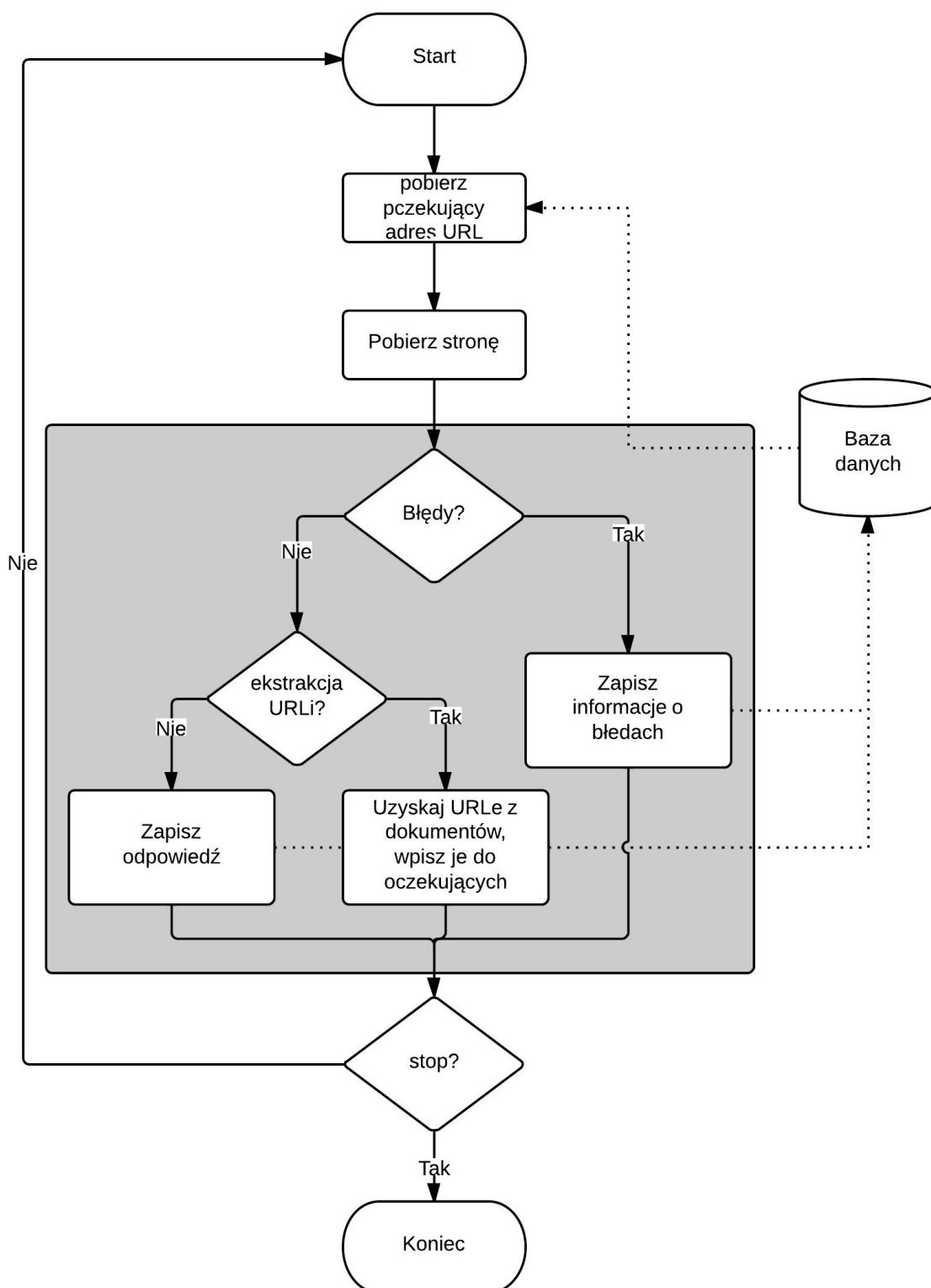
Pierwsza ścieżka przedstawiona jest na rysunku 4.2.2. Uruchamiany jest robot, który po odczytaniu konfiguracji i nawiązaniu połączenia z bazą danych zaczyna pobierać automatycznie dane z sieci. Liczba procesów wykonujących to zadanie nie ma narzuconego z góry ograniczenia, dane między procesami są synchronizowane na poziomie dostępu do bazy danych. Jak można zauważyć przedstawiony schemat nie różni się od podstawowej implementacji crawlera przedstawionej w rozdziale 2.

Rolę listy adresów URL przejęła baza danych. Jest to może rozwiązanie nieefektywne, ale proste w implementacji i wystarczające dla niniejszego projektu. W razie konieczności przyspieszenia działania robota można zastosować lekką bazę NoSQL, jak Redis, czy rozwiązanie cache'ujące, takie jak Memcached. W celu oszczędzenia czasu, nie zawsze następuje ekstrakcja adresów URL i dodawanie ich do bazy. Dzieje się to jedynie wtedy, gdy liczba adresów oczekujących jest mniejsza, niż czterokrotność liczby adresów odwiedzonych.

Druga ścieżka przepływu danych ma prostszy przebieg, dlatego ograniczone się jedynie do wypisania jej poszczególnych kroków.

1. Pobranie nieopublikowanego(flaga `opublikowana` ma wartość `FALSE`) rekordu z tabel `Zawartość`.
2. Parsowanie ciała strony za pomocą biblioteki Nokogiri.

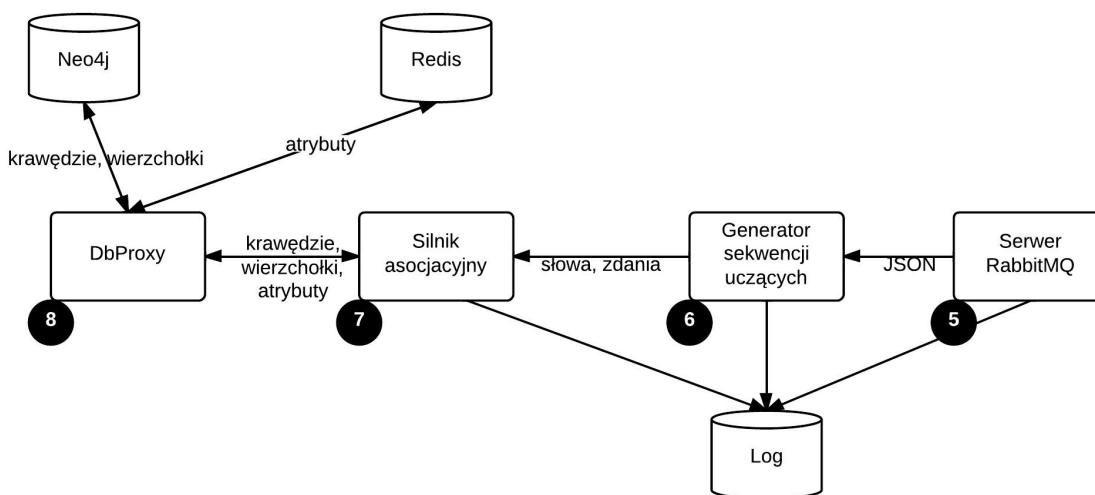
3. Pobranie z drzewa DOM interesujących elementów, np. całej zawartości umieszczonej między znacznikami HTML `<p>...</p>` i podzielenie jej na dwie grupy: pierwszą, która obowiązkowo musi zostać użyta do budowy grafu AGDS (np. tekst umieszczony w nagłówkach) i drugą, która zawiera przydatne, ale mniej wartościowe informacje - np. tekst z ciała artykułu.
4. Zapis tak przygotowanego tekstu do dwóch tablic, kolejno dla pierwszej i drugiej grupy.
5. Dodanie informacji o adresie URL przetwarzanej strony, zakodowanie jako JSON i opublikowanie poprzez kolejkę.



Rysunek 4.4: Schemat pierwszej ścieżki przepływu. Dla przejrzystości nie uwzględniono asynchronicznego sposobu pobierania stron.

4.3. Opis komponentów 5. - 8.

Na rysunku 4.3 przedstawiony jest detaliczny schemat reszty aplikacji. Jest to również kod pracujący autonomicznie, jedynym punktem wejścia jest serwer nasłuchujący na kolejce.



Rysunek 4.5: Detaliczny schemat komponentów 5. - 8. Podpisy na strzałkach odnoszą się do formy, jaką przyjmują przepływające przez aplikację dane.

4.3.1. Struktura przechowywanych informacji

Jak wspomniano wcześniej, baza grafowa cechuje się większą elastycznością, niż tradycyjne bazy relacyjne. Nie używa ona schematów, użytkownicy mają dowolność w kształtowaniu grafu i wielopoziomowych relacji łączących jego elementy.

4.3.2. Przepływ danych

Ten komponent posiada tylko jedną ścieżkę przepływu danych, związaną z tworzeniem struktury AGDS. Poniżej opisane są wszystkie przekształcenia, którym poddaje się dane odbierane przez serwer.

1. Odebranie danych przez serwer w postaci tekstowej, konwersja formatu JSON na obiekty natywne
2. Usunięcie nieprzydatnych ciągów znaków, np. charakterystycznego dla Wikipedii ciągu "[edit]" traktowanego przez parser HTML jak tekst, a niewnoszącego nowych informacji.
3. Podział tekstu, używając wyrażeń regularnych, na zdania i słowa.
4. Usunięcie zbyt krótkich słów, podział zbyt długich naturalnych zdań na krótsze (wydzielenie kontekstu).

5. Ujednolicenie kodowania(UTF-8), konwersja do małych liter.
6. (opcjonalny) Stemming poszczególnych słów.
7. Podanie tak utworzonej sekwencji uczącej na wejście silnika asocjacyjnego.
8. Zapis utworzonego grafu do baz danych.

Serwer oczekuje danych w postaci:

Listing 4.1:

```
1 { url: "http://www.example.com/", body: ["pierwsze zdanie", "drugie zdanie"] }  
2 //lub  
3 { url: "http://www.example.com/", body: { obligatory: ["Naglowek 1", "Naglowek 2"],  
    optional ["zdanie 1", "zdanie 2"] } }
```

5. Implementacja

Zanim struktura aplikacji zostanie omówiona w większym detalu, należy opisać platformę, na której powstawała. W związku z faktem, iż implementacja jest pewnego rodzaju eksperymentem, od którego nie wymaga się produkcyjnej sprawności, zaimplementowano ją w języku programowania Ruby. Pozwala on na szybkie wdrażanie nowych funkcjonalności, jest w pełni obiektywnym i bardzo elastycznym językiem programowania. Jednak ponieważ Ruby jest językiem interpretowanym i posiada dynamiczne typowanie, zaawansowane możliwości metaprogramowania raz zapewnia rozbudowane mechanizmy refleksji jego wydajność nie jest duża. W związku z charakterystyką wybranej platformy program ma szansę działać poprawnie jedynie dla systemów typu Unix(Linux, OS X). Zastosowane zostały dwie wersje interpreterów Ruby'ego: elementy 1. - 4. wymagają interpretera MRI, w wersji co najmniej 2.0.0. Jest to podstawowa implementacja tego języka, napisana w C, większość najważniejszych gemów(bibliotek) jest dostępna przede wszystkim dla tej dystrybucji. Natomiast moduły 5. - 8. wymagają użycia alternatywnej implementacji na JVM, o nazwie JRuby. Jest to wymaganie wykorzystywanej bazy grafowej Neo4j i dodatkowy powód, dla którego podzielono aplikację na dwie części.

Aplikacja używa 3 baz danych: PostgreSQL(<http://www.postgresql.org/>) do przechowywania ściągniętych stron, Neo4j(<http://www.neo4j.org/>) jako bazy grafowej i Redisa(<http://redis.io/>) jako lekkiej bazy klucz - wartość. Wszystkie posiadają biblioteki umożliwiające interakcję z nimi na poziomie obiektów języka, co znacznie ułatwia projektowanie aplikacji. Również każda z technologii używanych w projekcie jest technologią *Open Source*.

5.1. Wymagania instalacyjne

Aplikacja była rozwijana i testowana pod systemami typu Unix. Do instalacji i uruchomienia potrzebne są:

- system kontroli wersji Git (<http://git-scm.com/>),
- Ruby zainstalowany za pomocą systemu kontroli wersji(preferowany rbenv - <http://rbenv.org/>),
- Bundler (<http://bundler.io/>) i RubyGems (<http://rubygems.org/>),
- serwer PostgreSQL,

- baza grafowa Neo4j,
- serwer Redis,
- kolejka RabbitMQ.

Instrukcje instalacji, konfiguracji i uruchamiania poszczególnych komponentów znajdują się w dalszych częściach pracy.

5.2. Implementacja komponentów 1. - 4.

5.2.1. Konfiguracja

Konfiguracja możliwa jest poprzez dwa pliki znajdujące się w katalogu `./config`: `database.yml` i `config.yml`. Umożliwiają one dostarczenie potrzebnych informacji pozwalających na połączenie z relacyjną bazą danych, jak i na konfigurację zachowania aplikacji. Poniżej przedstawiony jest listing obu plików wraz z wyjaśnieniem dostępnych opcji.

`database.yml`

Listing 5.1:

```
1 defaults: &defaults
2   adapter: postgresql
3   encoding: unicode
4   pool: 5
5   username: user
6   password: pass
7   host: localhost
8
9 development:
10  database: taskmaster_dev
11  <<: *defaults
12 test:
13  database: taskmaster_test
14  <<: *defaults
```

Jest to plik instrumentujący używany w aplikacji ORM - *Active Record*, w celu umożliwienia połączenia z bazą. Konieczne podanie jest typu(*adapter*), użytkownika(*user*), hasła(*password*) i lokalizacji bazy(*host*). Zdefiniowano również specjalne środowisko testowe, dostępne pod kluczem `test`. Służy ono do definicji bazy powoływanej do egzystencji na czas testów, a następnie bezpowrotnie niszczonej.

config.yml

Listing 5.2:

```
1 crawler:
2   connections: 20
3   fetch_limit: 20
4   url_pattern: '\\/en\\.wikipedia\\.org\\/wiki\\/(?!\\/|[A-Za-z]+:)'
5   starting_page: 'http://en.wikipedia.org/wiki/Main_Page'
6 queue:
7   limit: 50
8 logger:
9   file: 'log/development.log'
10  enabled: true
```

Plik ten przechowuje informacje konfiguracyjne aplikacji. Kolejno, odpowiadają one za:

- crawler: connections mówi, ile jednoczesnych asynchronicznych połączeń może wykonywać jeden proces robota.
- crawler: fetch_limit określa, ile na raz URLi jest wysyłanych do robota w celu ściągnięcia z Internetu. Nie jest to jednoznaczne z parametrem connections, w razie wysłania większej ilości adresów URL, niż jest otwieranych połączeń crawler wykona kilka iteracji i zwróci dokumenty odpowiadające wszystkim adresom.
- crawler: url_pattern przechowuje wyrażenie regularne, używane do filtracji adresów URL pobieranych ze stron. Jedynie adresy zgodne z wyrażeniem są zapamiętywane w bazie.
- crawler: starting_page podaje stronę startową, od której należy rozpocząć przeglądanie sieci.
- queue: limit określa ile razy wywołana zostanie procedura przez kolejkę (ile stron zostanie wysłanych) przy jednym wywołaniu skryptu.
- logger: file przechowuje relatywną wobec folderu projektu ścieżkę logowania.
- logger: enabled to przełącznik, pozwalający na włączanie i wyłączanie loggera.

5.2.2. Instalacja i uruchomienie

Po pobraniu repozytorium i upewnieniu się, że pliki konfiguracyjne zawierają odpowiednie wartości, należy za pomocą *shella* wykonać w katalogu z projektem następujące komendy:

1. \$ bundle - powoduje ściągnięcie wszystkich zależności,
2. \$ rake db:create - tworzy bazę,
3. \$ PROJECT_ENV=test rake db:create - tworzy bazę testową,

4. `$ rake db:migrate` - migruje bazę do scheme'y wymaganej przez aplikację.

5. `$ PROJECT_ENV=test rake db:migrate` - migruje bazę testową.

6. (opcjonalnie) `$ rspec spec` w celu uruchomienia testów.

Aby uruchomić robota internetowego należy w katalogu projektu wykonać polecenie `$./download`. W celu uruchomienia klienta kolejki RabbitMQ wystarczy w katalogu projektu wywołać `$./publish`. Aplikacja udostępnia również konsolę, umożliwiającą programiście interakcję z załadowanym środowiskiem. Aby z niej korzystać należy w katalogu projektu wywołać polecenie `$./console`. Zmienna środowiskowa `PROJECT_ENV` używana jest do kontroli bazy danych, z którą łączy się aplikacja. Domyślnie przyjmuje ona wartość "development", a podczas testów "test". W celu uruchomienia np. konsoli z bazą testową, należy wywołać `$ PROJECT_ENV=test ./console`.

5.3. Implementacja komponentów 5. - 8.

5.3.1. Konfiguracja

Podobnie, jak opisana w sekcji 5.2.1 część projektu, również komponenty 5. - 8. są konfigurowane za pomocą pliku znajdującego się w katalogu `./config: config.yml`. Odpowiada on za przechowywanie informacji potrzebnych do połączenia z bazami, lokalizacji plików z logami itd., jak i do ustalania parametrów silnika asocjacyjnego. Część opcji dotyczy również silnika symulacyjnego umożliwiającego przeprowadzanie eksperymentów z sieciami ANAKG, jednak nie jest on bliżej omówiony w niniejszej pracy.

Listing 5.3:

```
1 payload:
2   optional_limit: 2000
3   min_word_length: 2
4   max_word_length: 25
5   max_sentence_length: 4
6
7 logger:
8   enabled: true
9   file: 'log/neo4ruby.log'
10
11 redis:
12   host: '127.0.0.1'
13   port: 6379
14
15 search_engine:
16   simulation:
17     alpha: 0.7
18     beta: 0.8
```

```
19  theta: 1.0
20  initial_exc: 0.95
21  max_iterations: 10
22  min_change_rate: 0.2
23
24  response_scanning:
25    limit: 5
26    levenshtein_max: 3
27    # stopwords after http://www.webconfs.com/stop-words.php
28    stopwords_file: 'data/stopwords'
29
30  answer_resolving:
31    limit: 10
32
33  experiment: 'expl'
```

Poszczególne opcje odpowiadają za:

- payload: optional_limit - maksymalna ilość słów, która jest wprowadzana do bazy grafowej z pojedynczej sekwencji uczącej (pojedynczej strony).
- payload: min_word_length - minimalna długość słowa (słowa krótsze są usuwane z sekwencji uczącej).
- payload: max_word_length - maksymalna długość słowa, j.w.
- payload: max_sentence_length - maksymalna długość zdania. Zdanie w aplikacji jest równoważne kontekstowi opisywanemu w rozdziale 3.
- logger: enabled - wł./wył. logowanie.
- logger: file - lokalizacja pliku z logami.
- redis: - dane potrzebne do połączenia z Redisem.
- search_engine: - parametry symulacji i asocjacyjnej wyszukiwarki internetowej.
- experiment: - nazwa eksperymentu. Jest to *de facto* ścieżka, w której przechowywane są pliki wbudowanej bazy grafowej. Zmieniając ją, zmienia się bazę, z którą łączy się aplikacja.

5.3.2. Instalacja i uruchomienie

Po pobraniu repozytorium i upewnieniu się, że pliki konfiguracyjne zawierają odpowiednie wartości, należy za pomocą *shell*a wykonać następujące komendy:

1. \$ bundle - powoduje ściągnięcie wszystkich zależności,
2. (opcjonalnie) \$ rake test w celu uruchomienia testów.

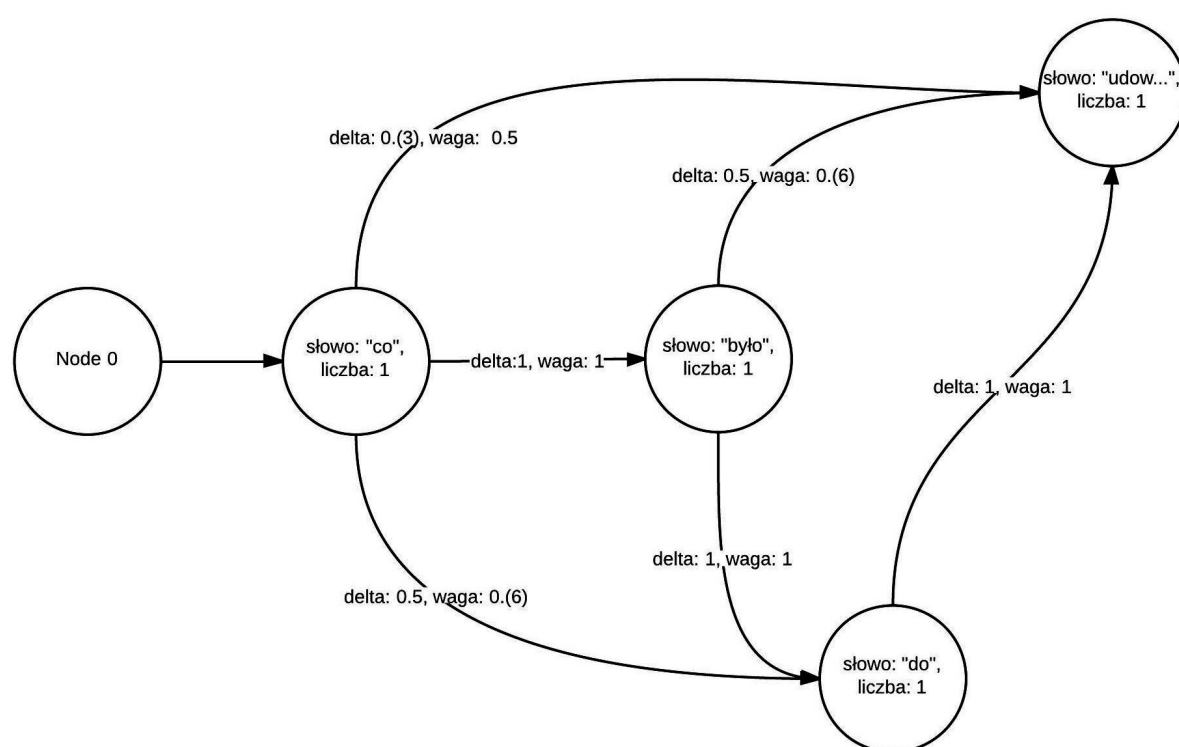
Aby uruchomić serwer należy będąc w katalogu aplikacji wywołać polecenie `$./start`. Przyjmuje ono dwa opcjonalne argumenty: `-e --experiment EXP_NAME` daje możliwość ręcznego ustawienia bazy, z którą łączy się aplikacja. Zmienna ustawiona w ten sposób nadpisze konfigurację zapisaną w pliku; `-q --queue QUEUE_NAME` pozwala na zmianę nazwy kolejki, na której nasłuchuje serwer. Należy jednak wspomnieć, iż ustawienia te były wykorzystywane głównie do rozwoju aplikacji na jej wczesnym stadium, docelowo wszystkie ustawienia zostaną przeniesione do pliku konfiguracyjnego.

Aplikacja posiada również skrypt ładujący całe środowisko i pozwalający na interaktywną jego eksplorację za pomocą linii komand. Uruchamiany jest on poleceniem `./console` z folderu projektu. W razie, gdy istnieje potrzeba połączenia się z inną bazą, niż wyszczególniona w pliku konfiguracyjnym, można użyć opcji `-e --experiment EXP_NAME`.

5.3.3. Struktura przechowywanych informacji

Z powodów wydajnościowych zdecydowano się na użycie dwóch rodzajów baz: Neo4j odpowiada za przechowywanie struktury grafu, jego trawersowanie i integralność. To rozwiązanie stosowane jest ze względu na przyjazne API i łatwość integracji z aplikacją. Jednak ze względu na dużą ilość wpisów do bazy podczas tworzenia struktury grafu AGDS (docelowo setki tysięcy stron) oraz konieczność sekwencyjnego budowania grafu, atrybuty elementów grafu, takie jak np. słowa wchodzące w skład sekwencji uczącej, czy wagi krawędzi są przechowywane przez prawie cały czas w pamięci programu i zapisywane okresowo w Redisie. Dzięki temu ograniczono znacząco ilość kosztownych operacji I/O.

Na rysunku 5.3.3 przedstawiona jest logiczna struktura grafu, widziana przez komponenty 5. - 7. Jest to struktura AGDS, powstała na podstawie sekwencji uczącej $S = \{co, bylo, do, udow...\}$. Widoczny jest również domyślny węzeł zerowy (Node 0), obecny w każdej strukturze Neo4j.



Rysunek 5.1: Przykład reprezentacji sekwencji uczącej w bazie grafowej.

6. Testy Oprogramowania

Rozdział ten jest poświęcony krótkiemu opisowi testów, którym poddano oprogramowanie w celu sprawdzenia poprawności działania. Skupiają się one głównie na poprawności tworzenia struktur AGDS oraz przytaczają wybrane dane dotyczące szybkości tworzenia tych struktur.

6.1. Testy poprawności tworzenia struktur AGDS

Podstawowe test działania oprogramowania opierają się na przykładach zamieszczonych w [6, s. 228 - 232]. Przedstawiono tam kompletny przykład tworzenia struktury AGDS i jej końcowy wygląd. Jeden z przykładów włączony został do zbioru testów automatycznych, zapewniając w ten sposób częste sprawdzanie najważniejszej funkcjonalności i umożliwiając szybkie wykrywanie błędów.

Test polega na zbudowaniu struktury AGDS modelującej przykładowe sekwencję. Sam proces budowy AGDS, jak i finalny wygląd sieci można znaleźć w [6, s.232].

Struktura tworzona jest na podstawie 5 sekwencji uczących, wg wzoru przedstawionego w rozdziale 3.

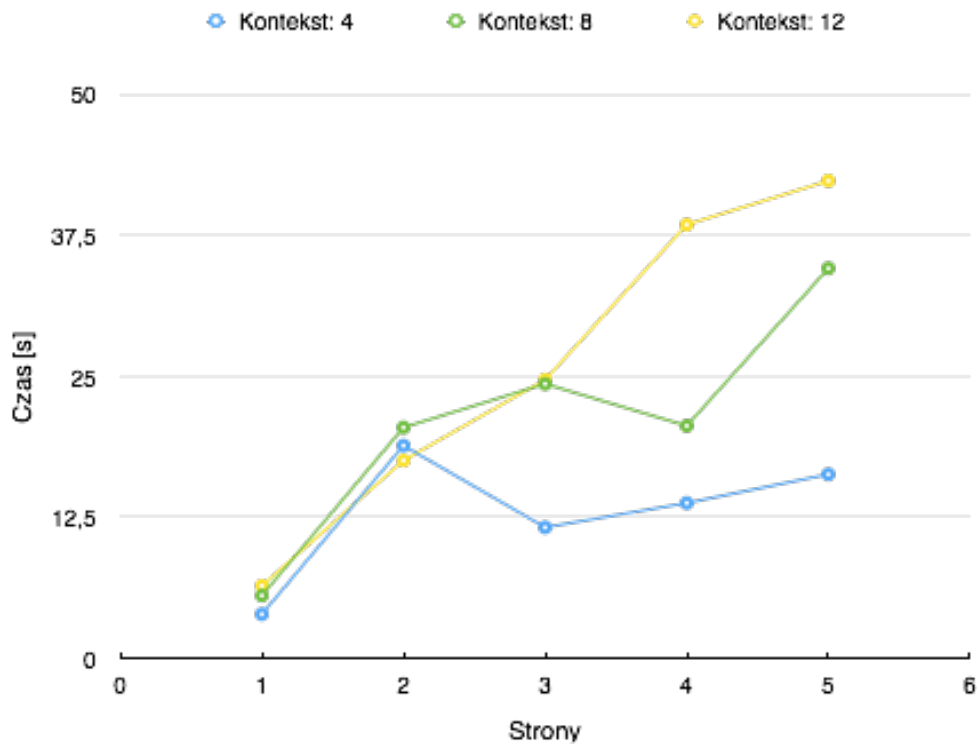
Sekwencje testowe: $S_1 = \{e1, e2, e3\}$, $S_2 = \{e4, e5, e2, e6\}$, $S_3 = \{e7, e5, e2, e8\}$, $S_4 = \{e7, e9, e8\}$, $S_5 = \{e4, e2, e3\}$.

Tabela 6.1 przedstawia konfrontację wyników wag krawędzi AGDS przedstawionych w [6, s. 232] i obliczonych przez program. Widać, iż niewielkie różnice wynikają z dokładności obliczeń numerycznych i nie są związane z niepoprawnością działania implementacji.

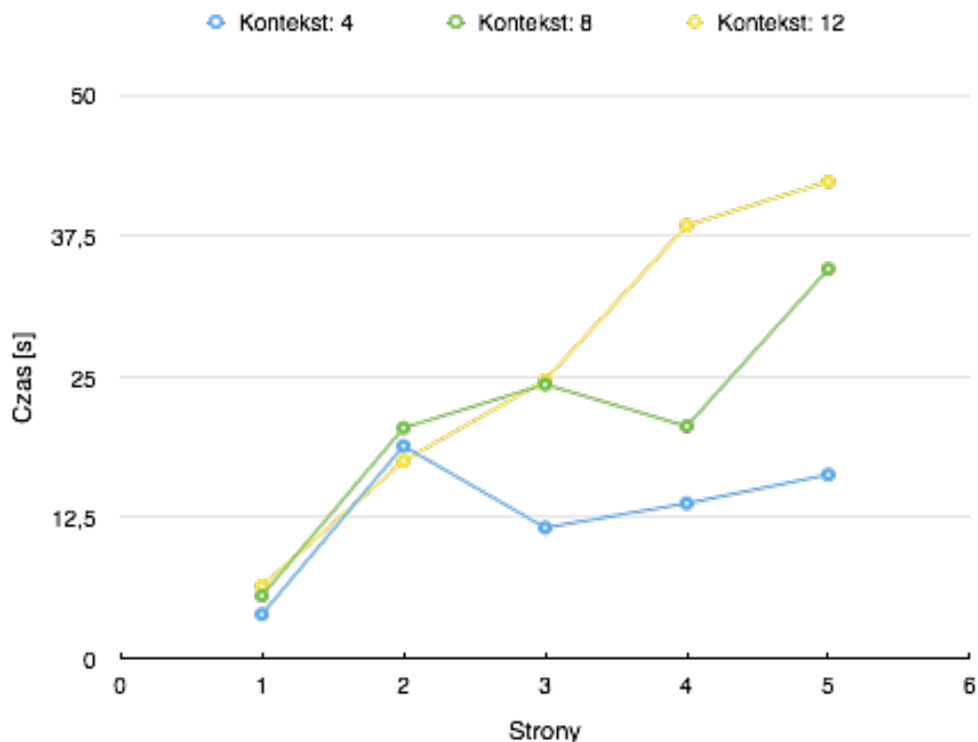
6.2. Inne testy i spostrzeżenia

Tabela 6.2 ilustruje czas rozbudowy grafu AGDS, w zależności od przyjętej długości kontekstu. Za każdym razem struktura AGDS budowana była od początku, z wykorzystaniem tej samej sekwencji uczącej, zmieniała się jedynie wartość pola konfiguracyjnego `max_sentence_length` (kolejno 4, 8 i 12).

Jak widać długość kontekstu ma znaczący wpływ na ilość krawędzi grafu, jak i na czas tworzenia struktury. Natomiast ilość węzłów pozostaje bez zmian, co jest zrozumiałe, ponieważ kontekst określa maksymalną długość połączenia między neuronami, natomiast nie ma wpływu na ich ilość.



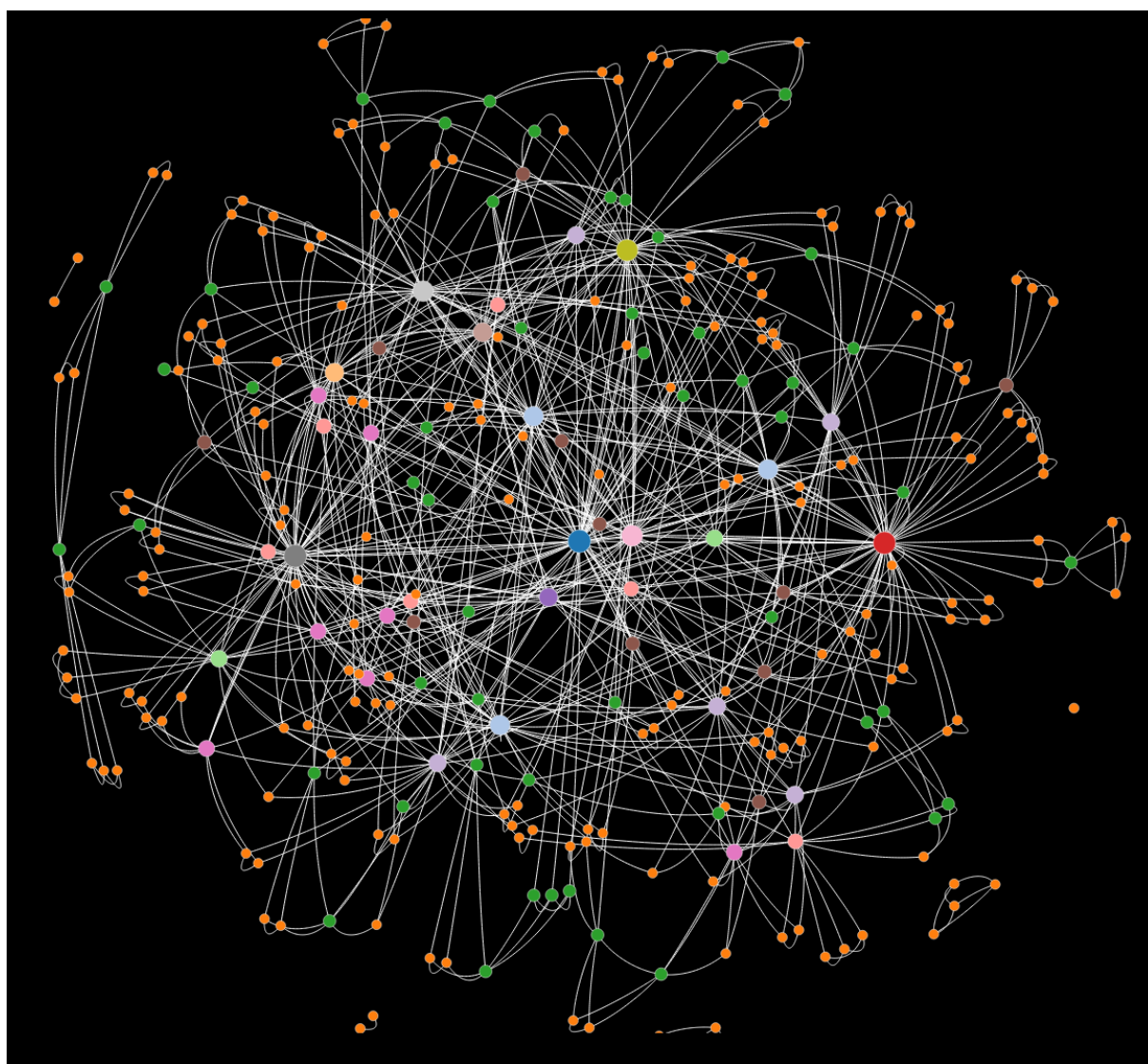
Rysunek 6.1: Ilustracja zmiany czasu zapisywania strony w zależności od długości kontekstu.



Rysunek 6.2: Ilustracja rosnącej liczby krawędzi w zależności od długości kontekstu.

Na rysunku 6.2 przedstawiona jest graficzna ilustracja struktury AGDS zbudowanej na podstawie artykułu znajdującego się pod adresem [http://en.wikipedia.org/wiki/Content_\(media\)](http://en.wikipedia.org/wiki/Content_(media)).

Składa się ona z 299 wierzchołków, i 858 krawędzi, kolor i wielkość wierzchołka odpowiadają częstotliwości występowania (i większy, tym częściej występuje). Wierzchołki o tym samym kolorze występują tą samą liczbą razy). Można zauważyć, iż w tekście znajduje się kilka słów, występujących często i posiadających wiele połączeń. Z analizy grafu dokonanej z poziomu aplikacji wynika, iż są to w większości słowa z listy *stopwords*, takie jak *the*, *an*, czy *of* (artykuł pisany jest w języku angielskim). Jednak figuruje wśród nich również słowo *content*, znajdujące się w tytule artykułu. Ciekawym zjawiskiem są podgrafy niepołączone z główną strukturą. Przedstawiają one prawdopodobnie definicje używające fachowego języka, który nie pojawia się w innych częściach artykułu.



Rysunek 6.3: Graficzna ilustracja grafu AGDS.

Tablica 6.1: Wyniki tworzenia struktury AGDS dla sekwencji testowych

Krawędzie	Waga teoretyczna	Waga wyliczona przez aplikację
Każda sekwencja jednokrotnie		
$e1 \rightsquigarrow e2$	1.0	1.0
$e1 \rightsquigarrow e3$	0.66	0.6666
$e2 \rightsquigarrow e3$	0.66	0.6666
$e2 \rightsquigarrow e6$	0.4	0.4
$e2 \rightsquigarrow e8$	0.4	0.4
$e4 \rightsquigarrow e2$	0.86	0.85714
$e4 \rightsquigarrow e3$	0.4	0.4
$e4 \rightsquigarrow e5$	0.66	0.6666
$e4 \rightsquigarrow e6$	0.29	0.2857
$e5 \rightsquigarrow e2$	1.0	1.0
$e5 \rightsquigarrow e6$	0.4	0.4
$e5 \rightsquigarrow e8$	0.4	0.4
$e7 \rightsquigarrow e2$	0.4	0.4
$e7 \rightsquigarrow e5$	0.66	0.6666
$e7 \rightsquigarrow e8$	0.59	0.5882
$e7 \rightsquigarrow e9$	0.66	0.6666
$e9 \rightsquigarrow e8$	1.0	1.0
$S_1, 5S_2, S_3, 3S_4, 2S_5$		
$e1 \rightsquigarrow e2$	1.0	1.0
$e1 \rightsquigarrow e3$	0.66	0.6666
$e2 \rightsquigarrow e3$	0.5	0.5
$e2 \rightsquigarrow e6$	0.71	0.7142
$e2 \rightsquigarrow e8$	0.2	0.4
$e4 \rightsquigarrow e2$	0.78	0.7826
$e4 \rightsquigarrow e3$	0.25	0.25
$e4 \rightsquigarrow e5$	0.83	0.8333
$e4 \rightsquigarrow e6$	0.38	0.3846
$e5 \rightsquigarrow e2$	1.0	1.0
$e5 \rightsquigarrow e6$	0.53	0.5882
$e5 \rightsquigarrow e8$	0.15	0.1538
$e7 \rightsquigarrow e2$	0.22	0.2222
$e7 \rightsquigarrow e5$	0.4	0.4
$e7 \rightsquigarrow e8$	0.63	0.6285
$e7 \rightsquigarrow e9$	0.86	0.8571
$e9 \rightsquigarrow e8$	1.0	1.0

Tablica 6.2: Czas tworzenia grafu AGDS w zależności od wielkości kontekstu

Początkowa ilość węzłów	Początkowa ilość krawędzi	Czas wstawiania pojedynczej strony[s]
Maksymalna długość kontekstu 4		
0	0	3.916
242	566	18.659
1681	8494	11.610
1705	8573	13.768
1915	9577	16.308
Maksymalna długość kontekstu 8		
0	0	5.57
242	1246	20.464
1681	12 622	24.323
1705	23 065	20.608
1915	29 167	34.566
Maksymalna długość kontekstu 12		
0	0	6.43
242	1854	17.539
1681	7493	24.698
1705	23 309	38.437
1915	33 137	42.324

7. Podsumowanie

W ramach projektu inżynierskiego zrealizowano aplikację zapisującą dokumenty pobrane z sieci za pomocą struktur asocjacyjnych AGDS.

Bibliografia

- [1] Carlos Castillo and Ricardo Baeza-Yates. *Web Crawling*. Universitat Pompeu Fabra. <http://grupoweb.upf.es/WRG/course/slides/crawling.pdf>.
- [2] Junghoo Cho and Hector Garcia Molina. Effective page refresh policies for web crawlers. *ACM Transactions on Database Systems*, 28(4), 2008.
- [3] John Gantz and David Reinsel. The digital universe decade – are you ready? <http://www.emc.com/collateral/analyst-reports/idc-digital-universe-are-you-ready.pdf>, 2010.
- [4] Padmini Srinivasan Gautam Pant and Filippo Menczer. *Crawling the Web*. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.62.4776&rep=rep1&type=pdf>.
- [5] Michael J.A. Berry Gordon S.Linoff. *Data Mining Techniques*. Wiley, third edition, 2011.
- [6] Adrian Horzyk. *Sztuczne systemy skojarzeniowe i asocjacyjna sztuczna inteligencja*. Akademicka Oficyna Wydawnicza Exit, Warszawa, 2013.
- [7] Jim Webber Ian Robinson and Emil Eifrem. *Graph Databases*. O'Reilly, 2013.
- [8] Aleksa Vukotic Jonas Partner and Nicki Watt. *Neo4j in Action*. Manning Publications, 2012.
- [9] Martijn Koster. *A Standard for Robot Exclusion*, 1994. <http://www.robotstxt.org/orig.html>.
- [10] Bing Liu. *Web Data Mining*. Springer, second edition, 2011.
- [11] Marian Boguna Santo Fortunato Alessandro Vespignani M. Angeles Serrano, Ana Maguitman. Decoding the structure of the www: facts versus sampling biases. *ACM Transactions on the Web*, 1(10), 2007.
- [12] Dariusz Mrozek. Wyszukiwanie pełnotekstowe. http://zti.polsl.pl/bdusm/BD3_FTS.pdf.

- [13] Brin S. and P. Lawrence. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.
- [14] Martin van den Berg Soumen Chakrabarti and Byron Dom. Focused crawling: a new approach to topic-specific web resource discovery. *Computer Networks*, 31(11-16):1623–1640, 1999.
- [15] Jonathan B. Spira. Information Overload: Now 900 Billion – What is Your Organization’s Exposure? <http://bit.ly/1coHvYF>, 2008.