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: *Szymon Sobański*
Kierunek studiów: *Automatyka i Robotyka*
Opiekun pracy: *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 czase 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. Implementacja 29
4.1. Wymogi i założenia 29
4.1.1. Budowa 29
4.1.2. Wejście/wyjście 30
4.1.3. Wymagania instalacyjne 31
4.2. Opis komponentów 1. - 4. 31
4.2.1. Konfiguracja 32
4.2.2. Instalacja i uruchomienie 33
4.2.3. Struktura przechowywanych informacji 34
4.2.4. Przepływ danych 35
4.3. Opis komponentów 5. - 8. 37
4.3.1. Konfiguracja 37
4.3.2. Instalacja i uruchomienie 39
4.3.3. Struktura przechowywanych informacji 39
5. Asocjacyjna wyszukiwarka internetowa 41
6. Podsumowanie 43

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, dotykający 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 zdobycie wiedzy o kosnumentach, łatwiejsze badanie rynku i dostosowanie oferowanych usług do potrzeb klientów [5, s. 1-26].

Głównym problemem zatem pozostaje kwestia sposobu porządkowania i filtracji dostępnych danych, w celu ich wykorzystania. Klasycznym przykładem aplikacji służacej 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. Przedstawione zostają argumenty przemawiające za takim rozwiązaniem, poddana dyskusji zostaje różnica pomiędzy proponowanym podejściem, a rozwiązaniami wiodącymi obecnie prym w zastosowaniach komercyjnych.

Ponadto zaproponowano wykorzystanie tak przechowywanych danych do budowy prostej asocjacyjnej wyszukiwarki internetowej, opartej o neuroasocjacyjne grafy wiedzy ANAKG[6, s. 223-244].

1.2. Zawartość pracy

W rodziale 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 **??** ma na celu opisanie problemów dotyczących ekstrakcji danych z dokumentu HTML. Zawiera on krótki opis formatu, przestawia jego zastowania i wyjaśnia, dlaczego poprawne uzyskiwanie informacji ze źródeł stron internetowych nie jest zadaniem trywialnym. Ponadto, zilustrowane są różne podejścia do parsowania zawartości dokumentów HTML, jak i przedstawiona jest pokrótce nomenklatura różnych typów stron, w kontekście wyszukiwarek internetowych.

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.

Następnie, w rozdziale 4, opisane są szczegóły implementacyjne aplikacji realizującej cele pracy. Wyjaśnione są szczegóły architektury, przytoczone zalety wybranch technologii, poruszony jest temat wydajności oprogramowania. Przedstawione jest również działanie całej aplikacji, wraz z wyjaśnieniem założeń testowych.

Rozdział 5 przedstawia aplikację wykorzystującą graf AGDS i zbudowaną w oparciu o niego sieć neuronową ANAKG do wyszukiwania stron internetowych. Przedstawia on podstawowe założenia stojące za budową sieci ANAKG, algorytm jej pobudzania, jak i sposób interpretacji wyników. Podjęta jest również próba ilustracji odpowiedzi sieci dla różnych parametrów symulacji.

2. Pozyskiwanie treści z Internetu

Pozyskiwaniem treści z sieci zajmuje się grupa oprogramowania nazywana sieciowymi pająkami(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 ciągłe odświeżanie posiadanych już informacji, poszukiwanie nowych stron i ew. eliminacja witryn, które juz nie funkcjonują. Wpływa to na charakterystykę działania pająków internetowych i oprogramowania z nimi współpracującego, które musi być w stanie stale przetwarzać dużą ilość informacji w krótkim czasie. W związku z ogromną ilością 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 błedów, unikanie "pułapek na pająkiźwiązanych z cyklami w strukturze sieci, czy przestrzeganie względów etycznych. Nie dziwi zatem pogląd założycieli firmy Google, którzy w jednym z artykułów stwierdzili, iż robot sieciowy jest najbardziej wyszukanym i najwrażliwszym 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 wszerz** mają za zadani 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 wgłab** [1] starają się odwiedzać strony odpowiadające w określony sposób potrzebom aplikacji. Sposób doboru stron może różnić się w zależności od zastosowań: od 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

Crawlery 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 do obrazu sieci niezgodnego ze stanem faktycznym.

2.1.4. Mirroring

Mirroring jest to sporzadzanie 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 zestam 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 spodzieać, 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 unikalnej listy adresów URL czekających na odwiedzenie (ang. *frontier*). W kazdej iteracji crawler zdejmuje pierwszy adres z listy i pobiera stronę znadjującą się pod tym adresem. Następnie następuje ekstrakcja adresów URL znajdujących sie 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 prawdopodne, ze względu na dużą ilość linków - średnio jest to dziesięć adresów URL na stronę[10, s.312].

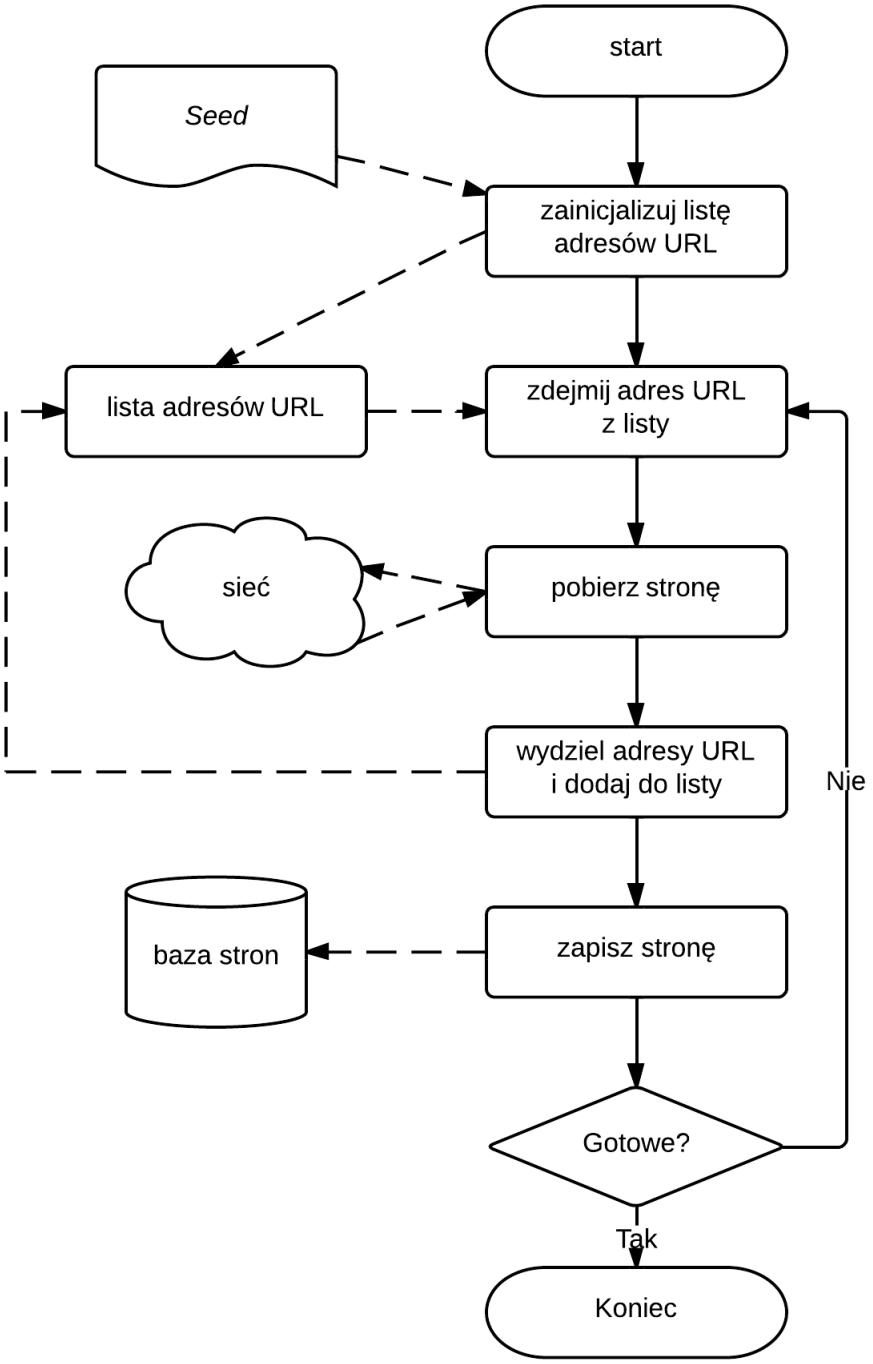
Dla robotów **przeszukujących wszerz** lista adresów URL implementowana jest zazwyczaj jako kolejka FIFO. Natomiast dla robotów **przeszukujące wgłąb** i **skupionych** używana jest kolejka priorytetowa, z priorytetem przypisanym na podstawie oceny przydatności danego adresu.

2.2.2. Uwagi implementacyjne

W związku z 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.

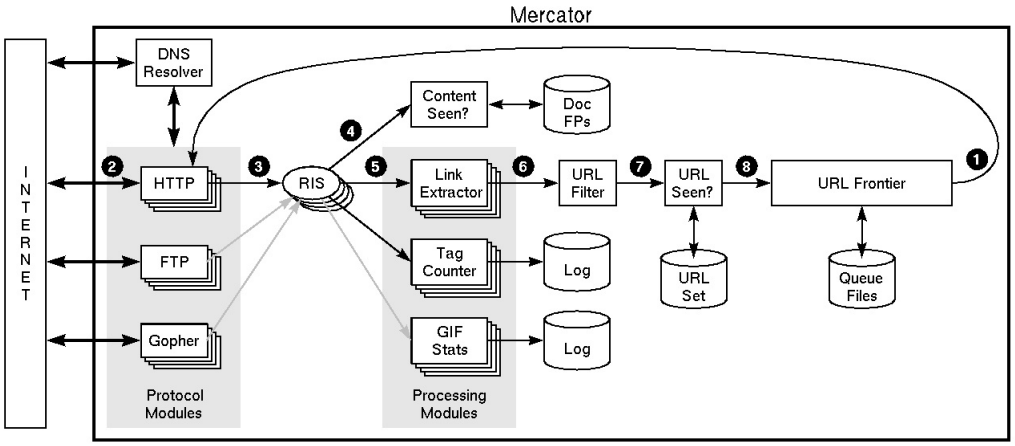
1. Stosowanie ścisłej kontroli nad ściąganiem stron: ograniczenie czasu trwania połączenia(stosowanie *timeoutów*), określenie górnej granicy ilości danych pobieranych z jednej strony(np. do 50 KB), wykrywanie nieskończonych pętli przekierowań, logowanie błędów(przekroczenie czasu połączenia, błędne kody odpowiedzi itd.).

2. Preprocessing pobranych źródeł, mający na celu eliminację częstych błędów występujących na stronach, takich jak np. niedomknięte 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



Rysunek 2.2: Schemat crawlera Mercator, implementującego m.in. cache DNS.

informację [12].

3. Doprowadzenie linków do postaci kanonicznej. Przede wszystkim należy zapewnić przedstawienie wszystkich URLi w postaci aboslutnej. Ponadto stosowane są heurystyki związane np. z usuwaniem numeru portu, o ile figuruje w danym adresie, pozbywaniem się nieznaczących sufiksów oznaczających typ dokumentu(np. .html), czy usuwanie tzw. fragmentu - części adresu następującej po znaku "#".

4. Unikanie tzw. pułapek na pająki (ang. *spide-traps*). *Pułapka na pająki* to strona, która generuje dynamicznie nieskończenie wiele adresów URL prowadzących do niej samej [4]. Przykładem takiej witryny jest aplikacja kalendarza, który generuje linki do stron przedstawiających poprzednie lub następne przedziały czasu. Aby uniknąć nieskończonej pętli stosuje się różnego rodzaju heurystyki, np. określa się górny limit ilości pobrań z danej domeny. Ma to jendak negatywny wpływ na aktualność informacji pobieranych ze stron o skończonej liczbe odnośników. Wpadnięcie wydajnego crawlera w taką pułapkę powoduje znaczne obciążenie serwerów i może być nawet interpretowane przez administratorów strony - pułapki jako atak denial of service [10, s. 322].

5. Wprowadzenie wielu wątków/procesów. Jak wspominano wcześniej, czas sekwencyjnego pobierania stron zależy w dużym stopniu od szybkości pobierania danych. Również, przy dużej ilości informacji, kosztowne może stać się również zapisaywanie danych na dysku. Jednym ze sposobów skrócenia czasu uzyskiwania nowych danych jest zastosowanie wielu równoległych wątków lub procesów, pobierających i przetwarzających dane niezależnie od siebie. Następuje wtedy jedynie konieczność synchronizacji dostępu i modyfikacji zarówno listy oczekujących do pobrania adresów URL. Takie podejście może w prosty sposób przyspieszyć działanie oprogramowania od 5 do 10 krotnie [10, s. 323].

   Innym ulepszeniem, które może zostać wprowadzone do wielowątkowej architektury, jest pobieranie danych w sposób asynchroniczny. Pozwala to na lepsze wykorzystanie dostępnej przepustowo-

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(dodanawanie nowych dokumentów, edycję i usuwanie istniejących), które w wypadku sieci postępują nadzwyczaj gwałtownie.

2.3.1. Świeżość i wiek

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

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

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

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

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 zminiejszenia liczby stron, które moga być nieaktualne. W tym modelu nie ma znaczenia historia 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 czase

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 (arykułów 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 iości 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 technki modelowania rozkładu prawdopodobieństwa zmian na stronie. Nie należy jednak zapominać, iż działalność prfesjonalnych, 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 infomracje 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, wpropwadzono protokół podstron wykluczonych z indeksowania *ang. robot exclusion protocol*. Przyjmuje on formę statycznej strony internetowej serwowanej przez lokalny dla adres URL `/robots.txt` [9]. Każda linia powinna mieć następujący format: `<pole>:<opcjonalnaspacja><wartosc><opcjonalnaspacja>`. Wg oficjalnej strony [9] `pole` może być zasąpione przez dwa ciągi znaków:

- `User-agent:` - ciąg znaków odnoszący się do wartość 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 odwiedziać.

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

```
# robots.txt for http://www.example.com/

User-agent: *
Disallow: /cyberworld/map/
Disallow: /tmp/ # these will soon disappear
Disallow: /bar.pdf
```

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

```
User-agent: cybermapper
Disallow:

# go away
User-agent: *
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 upłynąc 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 roszerzania kolekcji dokumentów o nowe pozycje. Rzadko jednak przechowywanie danych jest celem aplikacji. 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 reprezentatwyność 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 przedstawiancyh 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 prez 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 zminia 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 są aplikacje, dla których bazy grafowe są niezastapione, 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 kontatków, 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

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

3.2. Asocjacyjne grafowe struktury danych

Często **asocjacją** (skojarzeniem) nazywa się powiązanie ze sobą dwóch lub więcej zdefiniowanych wcześniej elementów. Zgodnie z [6, s. 47] tą definicję można rozszerzyć, opierając się na charakterystyce biologicznych organizmów żywych. Zauważono, iż niektóre organizmy mają umiejętność powiązywania dostepnych dla siebie danych w skomplikowane sieci zależnośc(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 kosstrukcji sztucznych systemów asocjacycjnych 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 międzyneuronalmy):

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

Definicja 2 *Powiązania łączące podobne i bliskie dane, ich kombinacje i układy nazywane są* ***powiązaniami 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* ***powiązaniami 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* ***powiązaniami asocjacyjnego kontekstu - ACON****.*

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

Definicja 5 *Bezpośrednie połączenia od receptorów lub neuronów przekazujących dane wejściowe do neuronu, który reporezentuje 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 wystepują 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 nasępne wierchołki,

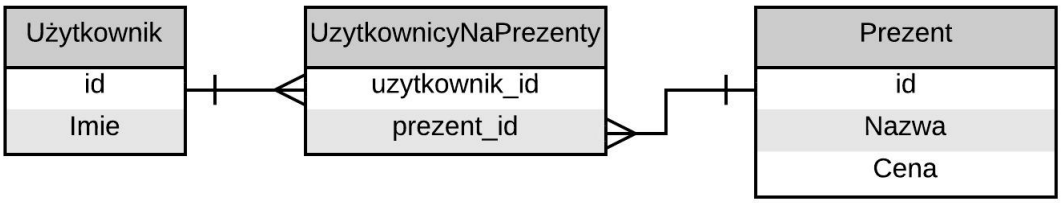
$EDEF$ - zbiór krawędzi dwustronnie skierowanych łączących wierzchołek definiujący z wierzchołkiem definiowanym w taki sposób, iż etykieta (waga) określa przejście od jednego, do drugiego wierzchołka.

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

Struktura AGDS jest statyczna i stego względu nie można za jej pomocą modelować innych relacji asocjacyjcnych przedstawionych w [6]. Typowo dane odwzorowywane są w niej za pomocą węzłów, natomiast krawędzie przedstaiają 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 zauważyc 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 *joinów*, 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. Trundo wprowadzać do takiej struktury zmiany, jest ona podatna na różne anomalie.

Tablica 3.2: Przykładowe dane

id	Imię
1	Mariusz
2	Weronika
3	Bazyl

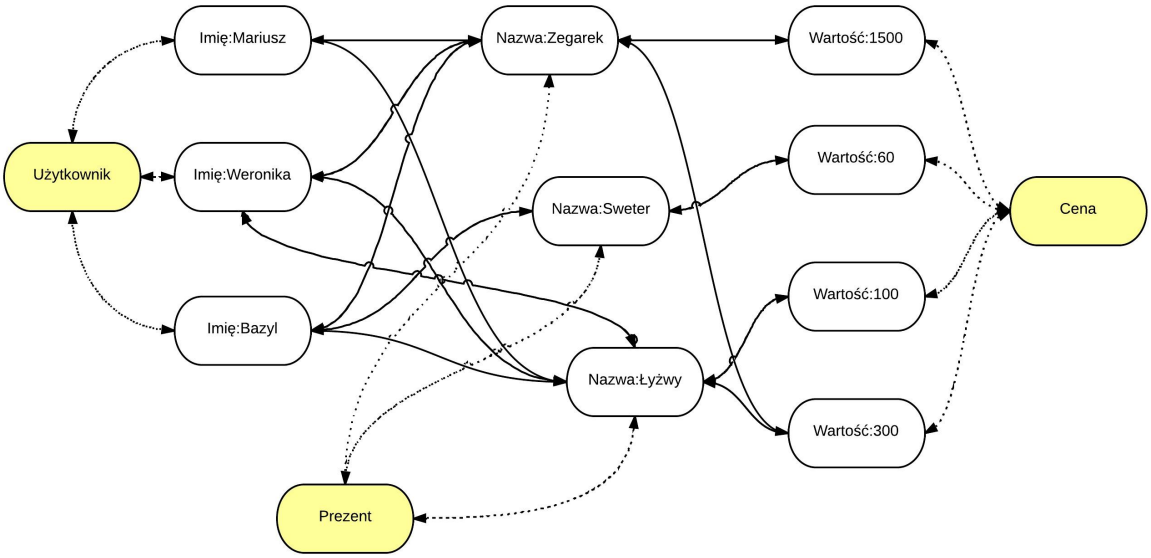
uzytkownik_id	prezent_id
1	1
1	2
2	2
2	3
3	4
3	1
3	5

id	Nazwa	Cena
1	Zegarek	1500
2	Łyżwy	300
3	Zegarek	300
4	Łyżwy	100
5	Sweter	60

Tablica 3.3: Tabela joinująca, zaprezentowana w celu zilustrowania danych

Imie	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 natychmiatowego otrzymania informacji o relacjach zachodzących między poszczególnymi obieketami. Otrzymanie infomracji o liscie 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 PostreSQLu, koszt wyszukiwania wynosi $O(logn)$. Wprawdzie można stosować index w postaci tablicy z haszowaniem, której średni czas dostepu 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 wprowadzic 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 wyszukiwannia 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**. Zazwczyaj AANG nie jest używane *per se*, ale dostoswywane 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 takch 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 częsci 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 reaktywe 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 fomułowana 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 specyficzej 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 rozpoczynana jest symulacja dyskretna z krokiem Δt,

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

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 płynącymmi od sąsiadujących neuronów. Na podstawie czasu, który upłynął między aktywacjami poszczególnych neuronów można określić tzw. **współczynnik aktywności**W połączenia.

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

$$\begin{cases} \delta^{act}_{SN,\widehat{SN}} = \delta^{act}_{SN,\widehat{SN}} + \frac{1}{\tau} \\ \tau = t_{postsyn} - t_{presyn} \end{cases} \tag{3.2}$$

Ze względu na sepcyfikę 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 neruonami. Ich wagi mogą być obliczone na podstawie zależności [6, s. 226]:

$$\begin{cases} w^{ACON}_{SN,\widehat{SN}} = \frac{2\delta^{act}_{SN,\widehat{SN}}}{\eta^{act}_{SN}+\delta^{act}_{SN,\widehat{SN}}} \\ \delta^{act}_{SN,\widehat{SN}} = \sum\limits_{(\rightsquigarrow ACON_\tau : SN \rightsquigarrow \cdots \rightsquigarrow \widehat{SN} \in AAT)} \frac{1}{\tau} \end{cases} \tag{3.3}$$

gdzie

η^{act}_{SN} - ilość aktywacji neuronu SN dla określonego zbioru sekwencji uczących $\mathbb{S}$.

$\delta^{act}_{SN,\widehat{SN}}$ - 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]:

$$\begin{aligned} exc^{SN}_t = NRF^{SQR}_{\alpha,\beta}(exc^{SN}_{t-1}, \theta^{SN}, x^t_1, \ldots, x^t_k) = \sum_{k=1}^{K} w_k x^t_k + \\ + \begin{cases} \alpha \cdot exc^{SN}_{t-1} + \frac{(\alpha-1)\cdot\beta\cdot(exc^{SN}_{t-1})^2}{\theta^{SN}} & exc^{SN}_{t-1} < 0 \\ \alpha \cdot exc^{SN}_{t-1} + \frac{(1-\alpha)\cdot\beta\cdot(exc^{SN}_{t-1})^2}{\theta^{SN}} & 0 \leq exc^{SN}_{t-1} < \theta^{SN} \\ -\beta \cdot \theta^{SN} & exc^{SN}_{t-1} \geq \theta^{SN} \end{cases} \end{aligned} \tag{3.4}$$

,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. Implementacja

W tym rozdziale opisana jest realizacja praktyczna projektu. Najpierw omówione zostaną wymagania stawiane całej aplikacji i omówione zostaną podstawowe założenia związane z jej implementacją. Nastepnie 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. Robota dostarczającego dane z sieci. Komponent musi mieć możliwość asynchronicznego pobierania stron WWW, udostępniać API do estrakcji adresów URL z ostatnio ściągniętych stron spełniających podane warunki oraz implementować *Robots Exclusion Protocol*.

2. Komponentu współpracującego z wyżej opisanym modułem, zapweniającego interfejs umożliwiający zapisywanie danych do zewnętrznej bazy. Umożliwiającego zapisaywanie 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 RabbiMQ, 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 zapewniajacą 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 strukutrę grafu w bazie grafowej, a dodatkowe informacje, takie jak atrybuty poszczególnych węzłów(np. częstość występowania), czy krawędzi(np. waga) przechowywać wlekkiej bazie klucz - wartość. Ważne jest ukreywanie szczegółów implementacyjnych, zwłaszcza wynikających z używania wielu rodzajów baz danych.

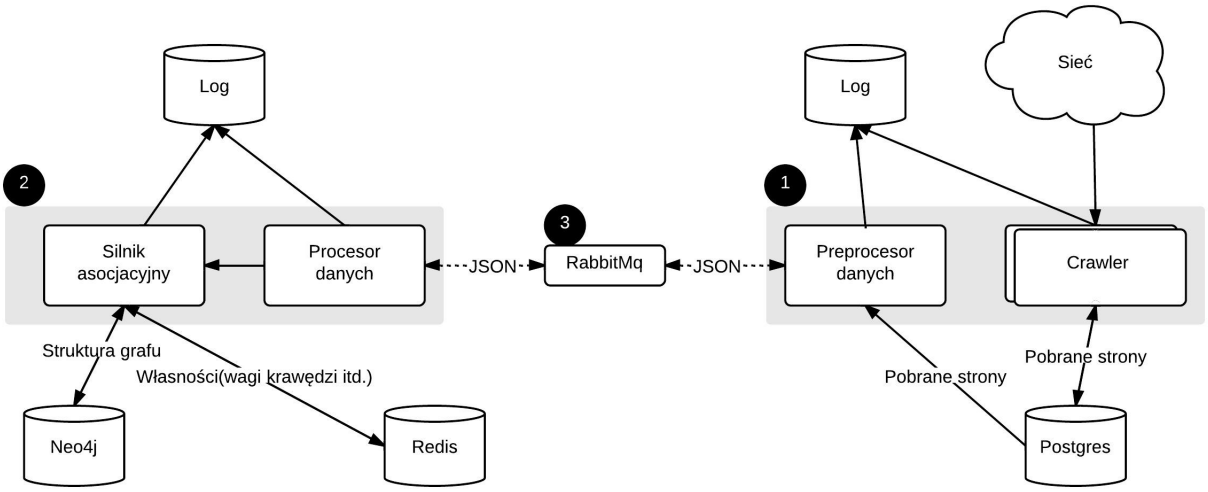
Zanim struktura aplikacji zostanie omówiona w większym detalu, należy opisać platformę, na której powstawała. W związku, z faktem, iż implementacja jest pewnego rodzaju eskperymentem, od którego nie wyamga się produkcyjnej sprawności, zaimplementowano ją w języku programowania Ruby. Pozwala on na szybką implementację nowych funkcjonalności, jest w pełni obiektowym i bardzo elastycznym językiem programowania. Jednak związku z tym, iż Ruby jest językiem interpretowanym i posiada dynamiczne typowanie, zaawansowane możliwości metaprogramowania i zapewnia rozbudowane mechanizmy refleksji jego wydajność nie jest duża. W związku z charakterystyką wybranej platformy program ma szanse działać poprawnie jedynie dla systemów typu Unix(Linux, OS X). Zstosowane zostały dwie wersje interepreterów Ruby'ego: elementy 1. - 4. wymagają interpretera MRI, wersji conajmniej 2.0.0. Jest to pierwsza 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 używanej bazy grafowej i jest głównym powodem użycia kolejki RabbitMQ do komunikacji między dwoma głównymi częściami aplikacji.

Aplikacja używa 3 baz danych: PostgreSQLa(`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 interakcje z nimi na poziomie obiektów języka, co znacznie ułatwia projektowanie aplikacji. Również każda z technologii używanych w projekcie jest technologią *Open Source*.

Każdy komponent posiada testy jednostkowe.

4.1.2. Wejście/wyjście

Aplikacja nie posiada interfejsu graficznego. Poszczególne komponenty są wzbudzane 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. Konkretne skrypty uruchamiające aplikacje, opcje ich wywoływania i struktura plików konfiguracyjnych opisana jest w dalszej części pracy.



Rysunek 4.1: Schemat architektury aplikacji. 1. - część napisana w MRI, 2. - część napisana w JRubym 3. - kolejka

4.1.3. Wymagania instalacyjne

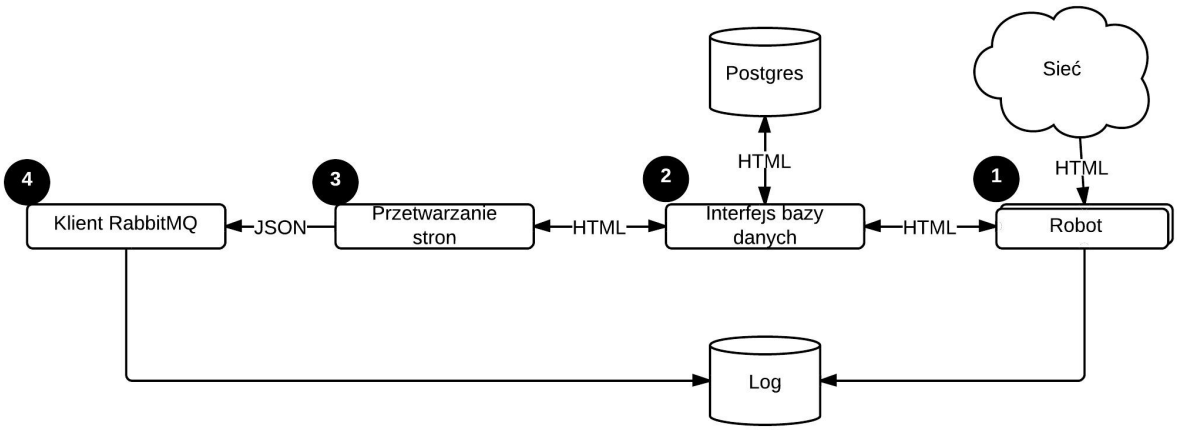
Aplikacja byla rozwijana i testowana pod systemami typu Unix. Do instalacji i uruchomienia potrzebne są:

- system kontroli wersji Git (`http://git-scm.com/`),
- Ruby zainstalowany za pomocą systemu kotroli wersji(preferowany rbenv - `http://rbenv.org/`),
- Bundler (`http://bundler.io/`) i RubyGems (`http://rubygems.org/`),
- serwer PostgreSQL,
- baza grafowa Neo4j,
- serwer Redis.

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



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. 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`

```
defaults: &defaults
  adapter: postgresql
  encoding: unicode
  pool: 5
  username: user
  password: pass
  host: localhost

development:
  database: taskmaster_dev
  <<: *defaults
test:
  database: taskmaster_test
  <<: *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 lokazlizacji

bazy(*host*). Zdefiniowano również specjalne środowisko testowe, dostępne pod luczem `test`. Służy ono do definicji bazy powoływanej do egzystencji na czas testów, a następnie bezpowrotnie niszczonej.

`config.yml`

```
crawler:
  connections: 20
  fetch_limit: 20
  url_pattern: '\/\/en\.wikipedia\.org\/wiki\/(?!\/|[A-Za-z]+:)'
  starting_page: 'http://en.wikipedia.org/wiki/Main_Page'
queue:
  limit: 50
logger:
  file: 'log/development.log'
  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 t 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 wyłącznie loggera.

4.2.2. Instalacja i uruchomienie

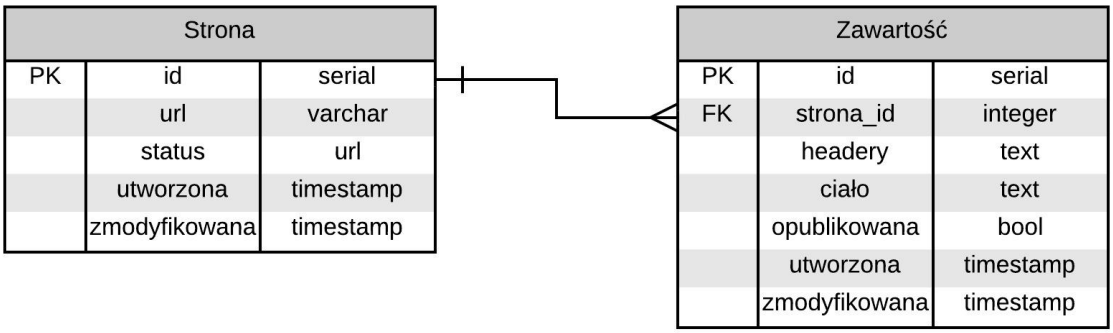
Po pobraniu repozytorum i upewnieniu się, że pliki konfiguracyjne zawierają odpowiednie wartości, należy za pomocą *shella* wykonać następujące komendy:

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 `PRJOJECT_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`.

4.2.3. Struktura przechowywanych informacji



Rysunek 4.3: Schema przechowująca informacje pobrane z sieci.

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: $oczekuje, pobierana, sukces, bd$, odpowiadające cyklowi pobierania i przetwarzania informacji. Wartość $pobierana$ ma na celu niedopuszczenie do pobierania tej samej strony przez dwa procesy 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 tegopola będzie TRUE, w przeciwnym razie FALSE.

4.2.4. Przepływ danych

Aplikacja posiada dwie niezależne ścieżki przepływu i przetwarzania danych:

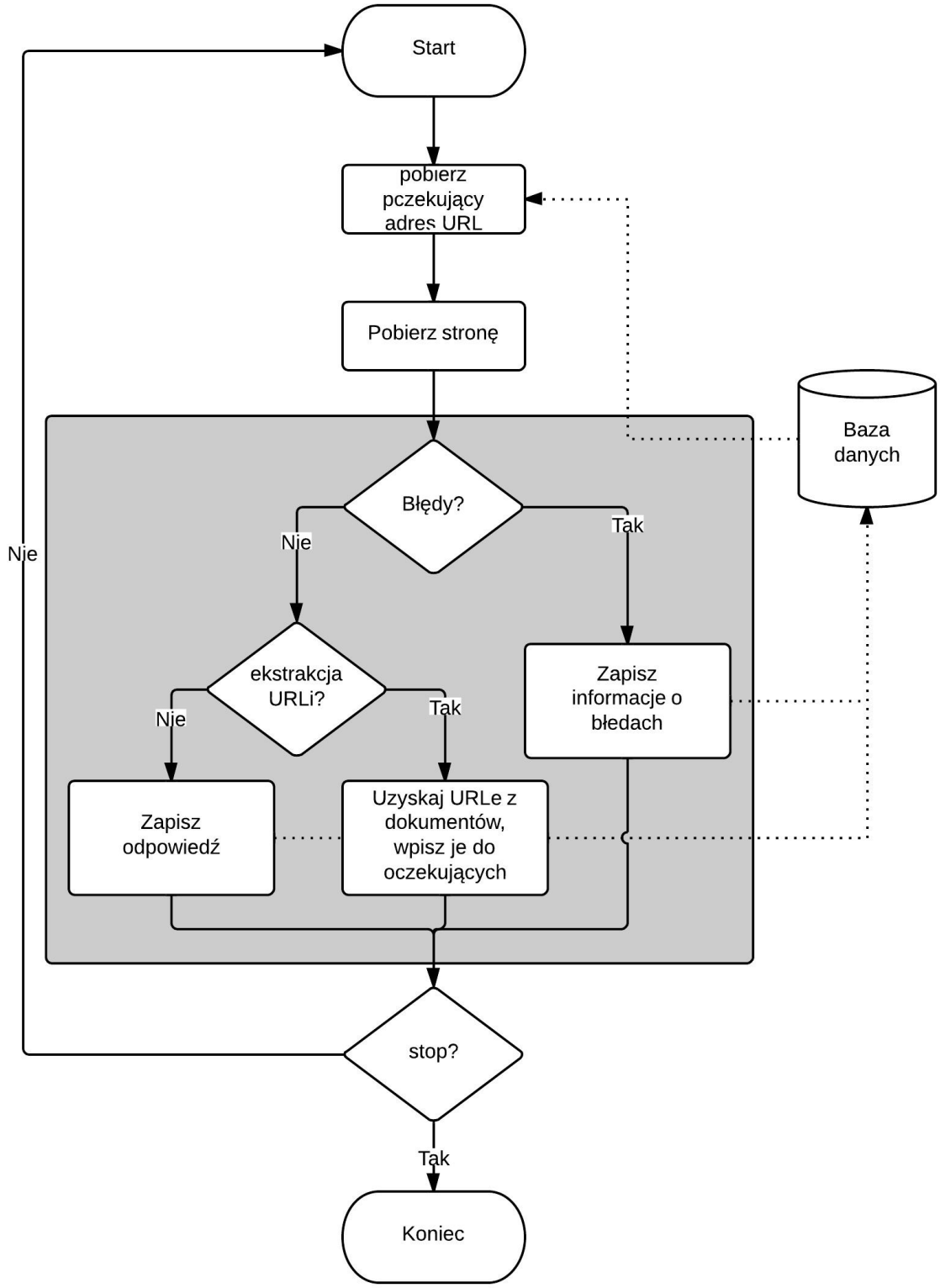
1. Ścieżka związana z pobieraniem danych z sieci.
2. Ścieżka przetwarzania ściągniętych danych i publikacja ich za pomocą kolejki.

Pierwsza ścieżka jest realizowana przez skrypt `download` i przedstawiona na rysunku 4.2.4. Uruchamia on robota, 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 przedstaionej 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. Nastepuje 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.

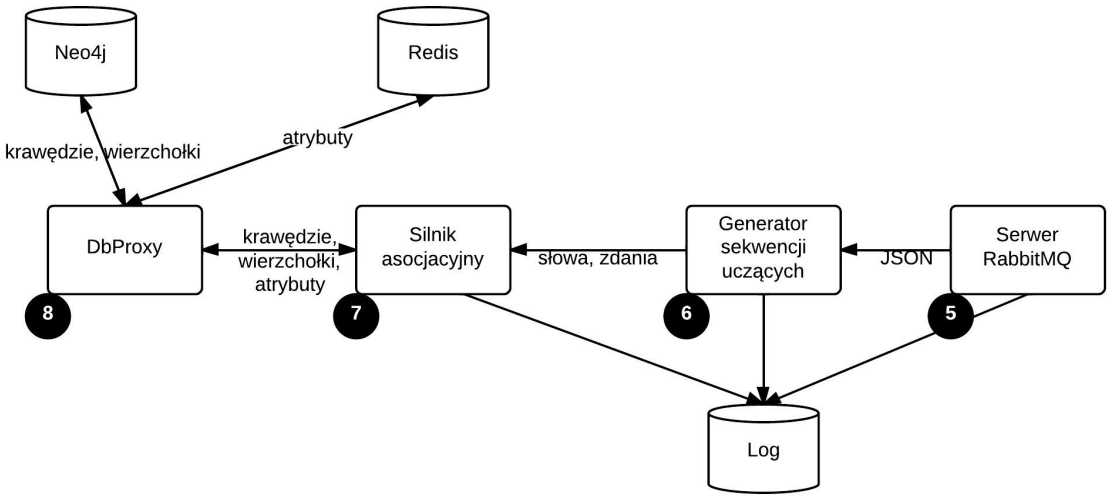
1. Pobranie nieopublikowanego(flaga `opublikowana` ma wartość FALSE) rekordu z tabel `Zawartość`.
2. Parsowanie ciała strony za pomocą biblioteki Nokogiri.
3. Pobranie z drzewa DOM interesujących elementów, np. całej zawartości umieszczonej między znacznikami HTML ... i podzielenie jej na dwie grupy: pierwszą, która obowiązkowo musi zostać użyta do budowy grafu AGDS(np. tekst umieszczony w nagłówkachi) i drugą, która zawiera przydatne, ale mniej wartościowe informacjue - np. tekst z ciała artyułu.
4. Podział tak przygotowanego tekstu na zdania i zapis 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 RabbitMQ.



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

4.3.1. Konfiguracja

Podobnie, jak opisana w sekcji 4.2.1 część projekt, również komponenty 5. - 8. są konfigurowane za pomocą pliku znajudującego się w katalogu `./config`: `config.yml`. Odpowiada on za przechowywanie informacji potrzebnych do połaczenia 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.

```
payload:
  optional_limit: 2000
  min_word_length: 2
  max_word_length: 25
  max_sentence_length: 4

logger:
  enabled: true
  file: 'log/neo4ruby.log'
```

```
redis:
  host: '127.0.0.1'
  port: 6379

search_engine:
  simulation:
    alpha: 0.7
    beta: 0.8
    theta: 1.0
    initial_exc: 0.95
    max_iterations: 10
    min_change_rate: 0.2

  response_scanning:
    limit: 5
    levenshtein_max: 3
    # stopwords after http://www.webconfs.com/stop-words.php
    stopwords_file: 'data/stopwords'

  answer_resolving:
    limit: 10

experiment: 'exp1'
```

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.

4.3.2. Instalacja i uruchomienie

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

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

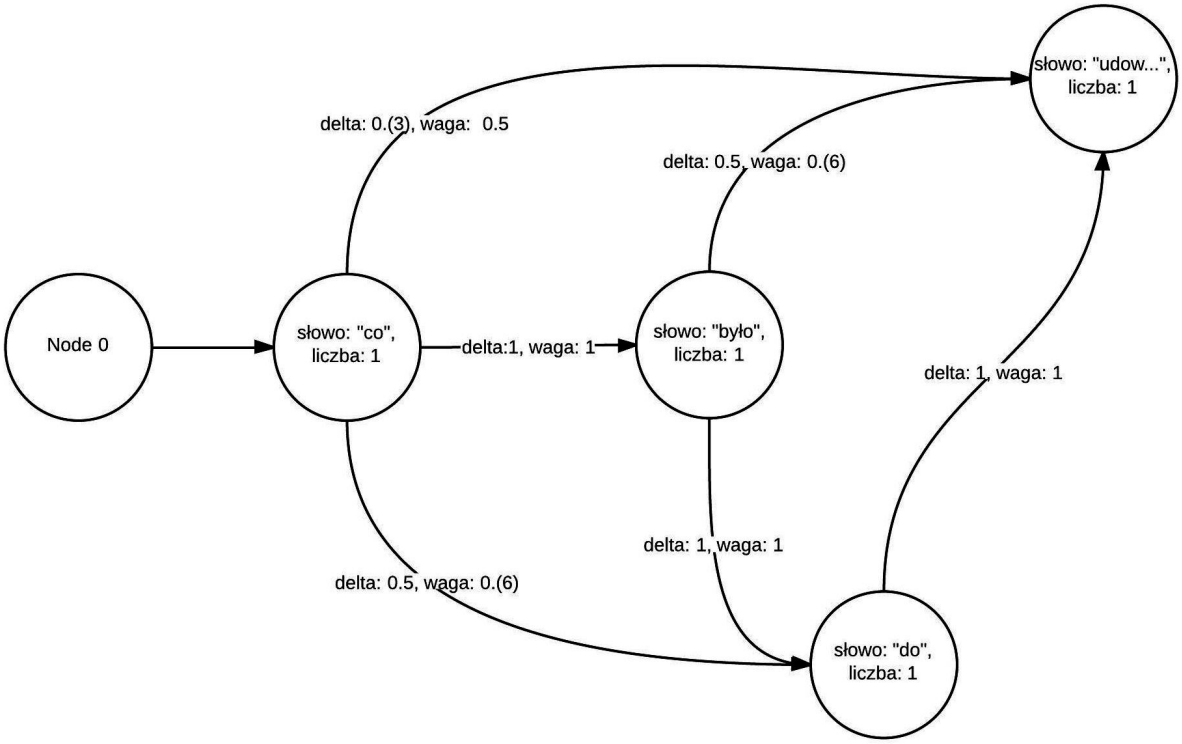
Aby uruchomić serwer należy będąc w katalogu aplikacji wywołać polecenie `$ ./start`. Przyjumje ono dwa opcjonalne argumenty: `-e --experiment EXP_NAME` daje możliwość ręcznego ustawienia bazy, z którą łączy się aplikacja. Zmienna ustawiona w ten sposób nadpisze konfigurację zapisaną w pliku; `-q --queue 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`.

4.3.3. 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. 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 wzgledu na duża 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 sa przechowywane przez prawie cały czas w pamięci programu i zapisywane okresowo w Redisie. Dzięki temu ograniczono znacząco ilość kosztownych operacji I/O.

Na rysunku 4.3.3 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 4.6: Przykład reprezentacji sekwencji uczącej w bazie grafowej.

5. Asocjacyjna wyszukiwarka internetowa

6. Podsumowanie

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 Wudawnicza 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.