Politechnika Poznańska
Wydział Informatyki
Instytut Informatyki

Streszczenie rozprawy doktorskiej

METODY AUTOMATYCZNEGO ROZSZERZANIA ONTOLOGII NA PODSTAWIE POWIĄZANYCH DANYCH

mgr inż. Jędrzej Potoniec

Promotor
prof. dr hab. inż. Joanna Józefowska
Promotor pomocniczy
dr inż. Agnieszka Ławrynowicz

Poznań, 2017

Rozdział 1

Wstęp

1.1 Motywacja

Semantyczny Internet (ang. *Semantic Web*) to pomysł przedstawiony początkowo w *Scientific American* przez twórcę Internetu jaki znamy dzisiaj, sir Tima Bernersa-Lee, wraz z Jamesem Hendlerem i Ora Lassila [4]. Główną ideą Semantycznego Internetu jest wymaganie, żeby zasoby internetowe były wzbogacone o wyrażone wprost znaczenie, oparte o zdefiniowane formalnie słownictwo ze współdzielonym znaczeniem. Stanowi to przeciwieństwo Internetu jak zwykle widzimy, gdzie semantyka jest domniemana, co czyni Internet nieprzyjaznym miejscem dla autonomicznych agentów niezdolnych do postrzegania świata jak ludzie.

Wskazane trudności nie mogły zostać przezwyciężone za pomocą istniejących technologii, utworzono zatem oddzielny zbiór standardów dla Semantycznego Internetu. *Resource Description Framework* (*RDF*) to standard definiujący reprezentację zasobów internetowych zrozumiałą dla maszyn, który skupia się wyłącznie na precyzyjnej reprezentacji informacji, całkowicie pomijając aspekt wizualny [30]. RDF jest wspierany przez język zapytań SPARQL (ang. *SPARQL Query Language*, w skrócie: SPARQL) i protokół SPARQL (ang. *SPARQL protocol*), będące standardowym sposobem odpytywania i pobierania danych w formacie RDF z Internetu [12]. Na nich oparty jest *Web Ontology Language OWL* mający na celu formalizację i współdzielenie semantyki słownictwa używanego w RDF, a także dostarczenie rozstrzygalnych algorytmów do wnioskowania dedukcyjnego zakorzenionych w logice.

Z rozpowszechnionego przyjęcia tych standardów, szczególnie RDF i SPARQL, wyłonił się nowy fenomen: obszar Internetu nazwany *Powiązanymi Danymi* (ang. *Linked Data*), utworzony specjalnie dla maszyn [3]. Otwarty podzbiór Powiązanych Danych, nazywany *Otwartymi Powiązanymi Danymi* (ang. *Linked Open Data*), powiększył się prawie 100-krotnie na przestrzeni ostatnich 10 lat, od 12 zbiorów w 2007 do 1139 zbiorów w 2017 [25]. Niestety, większość semantyki tych zbiorów danych jest dostępna tylko w formie tekstowych opisów na stronie internetowej, w artykule naukowym albo anotacjach w zbiorze danych.

W obecnej sytuacji podmiot publikujący zbiór Otwartych Powiązanych Danych ma mały zysk z utworzenia ontologii opisującej dane. Publikujący jest już zaznajomiony z danymi i nie potrzebuje ontologii, podczas gdy dostarczenie zgrubnego opisu w formie tekstowej dla wygody ludzkich użytkowników jest wystarczające. Dodatkowo uwzględniając, że inżynieria ontologii jest złożonym obszarem wymagającym zarówno wiedzy eksperckiej na temat danych jak i zasad modelowania ontologii, nie jest zaskakujące, że ontologie dla zbiorów Otwartych Powiązanych Danych są mało rozwinięte.

1.2 Problem badawczy

Tworzenie ontologii dla zadanego zbioru Powiązanych Danych byłoby dużo prostsze gdyby mogło zostać częściowo zautomatyzowane. W niniejszej pracy rozważa się następującą sytuację: dany jest zbiór Powiązanych Danych, to znaczy graf RDF dostępny za pomocą języka zapytań SPARQL. Jest on opisany ontologią, która może być zarówno bardzo płytka i składać się tylko ze słownictwa lub być bardzo złożoną ontologią bogatą w aksjomaty albo być na dowolnym etapie pomiędzy. W procesie tworzenia zbioru danych dużo wiedzy zostało w nim ukryte i rozważa się następujący problem: jak wspomóc inżyniera ontologii w odkrywaniu tej ukrytej wiedzy i użyć jej do rozszerzania ontologii o dodatkową, formalną semantykę. Celem nie jest zastąpienie inżyniera ontologii, ale raczej pomoc w rozwiązaniu rozpatrywanego zadania: sugerowanie nowych aksjomatów, które mogą być warte dodania do ontologii.

1.3 Cel i zakres pracy

W celu zdefiniowania pojęć w Rozdziale 2 rozprawy wprowadzone są formalizmy i technologie używane w Semantycznym Internecie: Resource Description Framework, język SPAQRQL, Logiki Deskrypcyjne (ang. *Description Logics*) i OWL 2 Web Ontology Language. Rozdział 3 rozprawy przedstawia istniejące prace innych autorów w powiązanych obszarach badawczych, w szczególności: uczenie się ontologii z tekstu w języku naturalnym, uczenie się ontologii w wyniku interakcji z użytkownikiem, metody oparte na Indukcyjnym Programowaniu w Logice (ang. *Inductive Logic Programming*), uczenie się ontologii z grafów RDF dostępnych lokalnie oraz z Powiązanych Danych.

Poczynając od Rozdziału 4 rozprawy, przedstawiony jest nowy wkład w obszar badawczy, poczynając od metody odkrywania pojawiających się podklas, które jeszcze nie zostały nazwane, ale są wspierane przez dane: przedstawiony jest algorytm Fr-ONT-Qu, którego fragmenty metoda wykorzystuje, dyskutowany jest model matematyczny służący do wyboru optymalnego zbioru częstych wzorców oraz sposób zamieniania ich na aksjomaty, a następnie przedstawiona jest implementacja zaproponowanej metody.

W Rozdziale 5 rozprawy przedstawiony jest sposób rozszerzenia algorytmu eksploracji atrybutów, pochodzącego z Formalnej Analizy Pojęć (ang. *Formal Concept Analysis*), o wykorzystanie algorytmów uczenia maszynowego do wspomagania użytkownika: przedstawione jest wprowadzenie do Formalnej Analizy Pojęć i algorytmu eksploracji atrybutów, dyskutowany jest sposób rozszerzenia algorytmu i formalizowany jest rozpatrywany problem klasyfikacji, prezentowana jest również implementacja zaproponowanego rozwiązania.

Rozdział 6 rozprawy przedstawia Swift Linked Data Miner, algorytm odkrywania wzorców w Powiązanych Danych: dyskutowany jest sposób efektywnego pobierania danych z grafu RDF i ich efektywna organizacja w pamięci, następnie rozpatrywane zadanie jest formalizowane jako problem odkrywania częstych wzorców i przedstawione są algorytmy go rozwiązuje. W dalszej części zaprezentowane są wyniki eksperymentu crowdsourcingowego, mającego za zadanie ocenić poprawność metody oraz wyniki eksperymentów obliczeniowych, służących ocenie efektywności algorytmu.

W Rozdziale 7 rozprawy znajduje się podsumowanie i porównanie zaproponowanych metod oraz zarysowane są możliwe zastosowania i dalsze kierunki prac.

Rozdział 2

Semantyczny Internet

2.1 Resource Description Framework

Resource Description Framework (RDF) to język umożliwiający grafową reprezentację zasobów Internetowych [30]. Podstawowym pojęciem w RDF jest *trójka RDF* (ang. *RDF triple*), która składa się z *podmiotu* (ang. *subject*), *predykatu* (ang. *predicate*) i *dopełnienia* (ang. *object*). Zbiór trójek RDF jest nazywany *grafem RDF* (ang. *RDF graph*), a podmioty i dopełnienia trójek grafu tworzą zbiór węzłów RDF (ang. *RDF nodes*). W niniejszej pracy RDF jest przedstawiany w składni *Turtle* [9].

Każda część trójki musi być jednym z termów RDF: *Międzynarodowym Identyfikatorem Zasobu* (ang. *International Resource Identifier (IRI)*), węzłem anonimowym (ang. *blank node*) albo literałem (ang. *literal*). IRI to globalny identyfikator o składni zdefiniowanej w RFC3987 [10]. Węzeł anonimowy to również identyfikator zasobu, ale mający zasięg pojedynczego grafu RDF. Literał opisuje pojedynczą daną i składa się z konkretnej wartości, np. łańcucha znaków, oraz jej typu danych (ang. *datatype*).

2.2 Język zapytań SPARQL

Zbiór termów dopuszczalnych w zapytaniach SPARQL składa się z termów RDF oraz zmiennych [12]. Zmienna składa się z dowolnej nazwy poprzedzonej przez `?` lub `$`. Węzeł anonimowy w zapytaniu SPARQL również służy za zmienną.

Wzorzec trójki (ang. *triple pattern*) to trójka RDF, w której dowolna liczba części została zastąpiona przez zmienne. *Prosty wzorzec grafowy* (ang. *basic graph pattern (BGP)*) to zbiór wzorców trójek traktowany jako koniunkcja. Wzorce trójkowe w SPARQL są wyrażane w składni Turtle. Wyrażenie `FILTER(filtr)` pozwala na dodatkowe ograniczenie rozwiązań dla BGP: jeżeli, dla danego rozwiązania, wartość logiczna wyrażenia `filtr` zostanie obliczona jako fałsz albo nie uda się jej obliczyć, rozwiązanie jest odrzucane.

Podstawową formą zapytań SPARQL SELECT jest `SELECT glowa WHERE { wzorzec }`, gdzie `wzorzec` to BGP, opcjonalnie uzupełniony o wyrażenia `FILTER`, podczas gdy `glowa` definiuje projekcję rozwiązań dla wzorca. Baza danych zawierająca graf RDF jest nazywana *magazynem RDF* (ang. *RDF store*), a usługa służąca do odpowiadania na zapytania SPARQL jest nazywana *końcówką SPARQL* (ang. *SPARQL endpoint*) i zwykle jest identyfikowana przez adres URL.

2.3 Powiązane Dane

Powiązane Dane (ang. *Linked Data*) to pojęcie stworzone przez sir Tima Bernersa-Lee w 2006 w celu wskazania, że Semantyczny Internet to nie samo umieszczanie danych w Internecie, ale raczej łączenie ich w *Sieć Danych* (ang. *Web of Data*), żeby ludzie i maszyny eksplorujące fragment danych były w stanie odkryć inne istotne fragmenty [3]. Na początku 2007 okazał się, że dane powinny być publikowane na otwartych licencjach, które umożliwiają ich ponowne użycie za darmo i w ten sposób powstało pojęcie *Otwartych Powiązanych Danych* (ang. *Linked Open Data (LOD)*) [5]. Metody przedstawione w rozprawie nie zależą od otwartości, ale zależą od dostępności danych w końcówce SPARQL.

DBpedia [16] odgrywa kluczową rolę w LOD, jako że dostarcza informacji na bardzo szeroki zakres tematów, dzięki czemu publikujący dane może, stosunkowo łatwo i niezależnie od dziedziny swoich danych, połączyć je z *DBpedią*.

2.4 Logiki Deskrypcyjne

Logiki Deskrypcyjne (ang. *Description Logics (DLs)*) to formalizm reprezentacji wiedzy odpowiedni do opisu *uniwersum dyskursu* (ang. *universe of discourse*) przez zebranie najpierw zbioru adekwatnych pojęć służących do opisu, a następnie użycia ich do opisania właściwości obiektów w uniwersum. W szczególności interesująca jest logika $\mathcal{EL}^{++}$ [2], stanowiąca podbudowę dla języka *OWL 2 EL* [18]. Rozprawa przedstawia definicję pojęć i konstrukcji wykorzystywanych w tej logice, a także wprowadza podstawowe problemy wnioskowania.

2.5 OWL 2 Web Ontology Language

OWL 2 Web Ontology Language (*OWL 2*) to standard organizacji W3C służący modelowaniu i wymienianiu ontologii, tj. modeli konceptualnych, na potrzeby Semantycznego Internetu. Ontologia OWL 2 może być przedstawiana jako graf RDF i jest w związku z tym zwykle zapisywana używając jednego z zapisów RDF. OWL 2 jest wyposażony w formalnie zdefiniowaną semantykę opartą na Logikach Deskrypcyjnych. Wyróżnia się trzy profile języka: OWL 2 RL, OWL 2 QL, OWL 2 EL. OWL 2 EL, oparty na Logice Deskrypcyjnej $\mathcal{EL}^{++}$ jest szczególnie odpowiedni do wyrażania dużych ontologii ze względu na dostępne konstrukcje języka, a także ze względu na możliwość wnioskowania w PTIME [18]. Ponieważ ontologia stworzona półautomatycznie może łatwo stać się bardzo duża, jest zasadnym wybranie profilu, odpowiedniego do dużych ontologii.

Słownictwo OWL 2 EL składa się z następujących zbiorów: *klas* (ang. *classes*), *własności obiektowych* (ang. *object properties*), *własności danych* (ang. *data properties*), *indywiduów* (ang. *individuals*), *typów danych* (ang. *datatypes*), *literałów* (ang. *literals*) i faset (ang. *facets*). W OWL 2 EL dozwolone jest korzystanie z 19 typów danych. Ontologia OWL 2 EL to zbiór aksjomatów, opcjonalnie wzbogacony o anotacje i polecenia importu. Z punktu widzenia przedstawianych metod istotne są wyłącznie aksjomaty. W rozprawie przedstawiona jest pełna lista dozwolonych w OWL 2 EL zakresów danych (ang. *data ranges*), wyrażeń klasowych (ang. *class expressions*) i aksjomatów, wraz z odpowiadającą im semantyką zgodnie ze specyfikacją [11].

Rozdział 3

Modelowanie matematyczne w uczeniu się ontologii

W niniejszym rozdziale rozważa się problem uczenia się ontologii, w którym celem jest podział istniejącej klasy w ontologii na zbiór podklas takich, że podklasy pokrywają tak dużo indywiduów należących do klasy jak to możliwe przy jednoczesnym zachowaniu przecięć między podklasami tak małym jak to możliwe. Ponadto dąży się do uzyskania formalnych definicji podklas poprzez użycie równoważności, tj. `owl:equivalentClass` i słownictwa już występującego w ontologii.

3.1 Algorytm Fr-ONT-Qu

Fr-ONT-Qu to algorytm odkrywania wzorców z grafu RDF dostępnego w końcówce SPARQL [15]. Wzorce odkryte przez Fr-ONT-Qu są wyrażane jako zapytania SPARQL SELECT z pojedynczą zmienną w głowie i zawierają wyłacznie klauzulę WHERE, która składa się jedynie z BGP i opcjonalnych wyrażeń filtrujących. Ze względu na naturę algorytmu, graf odpowiadający BGP jest drzewem ukorzenionym w zmiennej z głowy zapytania, a wszystkie wyrażenia filtrujące są postaci *zmienna* `>=` *liczba* bądź *zmienna* `<=` *liczba*. Fr-ONT-Qu wykorzystuje przeszukiwanie *najpierw najlepszy* (ang. *best-first search*), żeby iteracyjnie ulepszać wzorce.

3.2 Model matematyczny

Niech $\mathbb{P} = \{P_1, P_2, \ldots, P_n\}$ będzie zbiorem wzorców odkrytych przez Fr-ONT-Qu. Niech `C` będzie klasą, która ma zostać rozszerzona o nowe podklasy. Niech $\mathbb{I} = \{I_1, I_2, \ldots, I_m\}$ będzie zbiorem IRI, które należą do klasy `C` zgodnie z rozważaną końcówką SPARQL. Niech $\mathbb{R}(P)$ będzie zbiorem odpowiedzi na zapytanie $P \in \mathbb{P}$.

Celem jest wybór podzbioru $\mathbb{X} = \{P_{i_1}, \ldots, P_{i_k}\}$ z $\mathbb{P}$ takiego, że liczba IRI z $\mathbb{I}$ obecnych w wynikach dokładnie jednego z zapytań w $\mathbb{X}$ była maksymalizowana. Łatwo zaobserwować, że bez dodatkowych ograniczeń, przedstawiony problem może mieć trywailne rozwiązanie, w którym $\mathbb{X}$ zawiera jeden, bardzo szeroki wzorzec. Żeby temu zapobiec wprowadza się parameter λ, który oznacza mninimalny rozmiar $\mathbb{X}$. Powyższy opis przedstawia problem uczenia się ontologii jako problem optymalizacyjny. Żeby go rozwiązać, proponuje się zastosowanie zero-jedynkowego programowania liniowego (ang. *binary linear programming*).

Relacja pomiędzy $\mathbb{P}$ i $\mathbb{I}$ jest zdefiniowana przez macierz A:

$$A_{i,j} = \begin{cases} 1 & I_i \in \mathbb{R}(P_j) \\ 0 & \text{w przeciwnym przypadku} \end{cases} \qquad \forall i \in \{1, 2, \ldots, m\}\, \forall j \in \{1, 2, \ldots, n\} \tag{3.1}$$

Ponadto, przez M rozumie się bardzo dużą liczbę względem rozmiaru problemu.

Model wykorzystuje trzy grupy zmiennych. Po pierwsze, zmienne x_j definiują rozwiązanie, tj. zbiór $\mathbb{X}$:

$$x_j = \begin{cases} 1 & P_j \in \mathbb{X} \\ 0 & \text{w przeciwnym przpadku} \end{cases} \qquad \forall j \in \{1, 2, \ldots, n\} \tag{3.2}$$

Zbiór zmiennych y_i definiuje zbiór IRI z $\mathbb{I}$ takich, że każdy z tych IRI jest obecny w wynikach przynajmniej jednego z zapytań z $\mathbb{X}$:

$$y_i = \begin{cases} 1 & \exists j \ (P_j \in \mathbb{X} \wedge I_i \in \mathbb{R}(P_j)) \\ 0 & \text{w przeciwnym przypadku} \end{cases} \qquad \forall i \in \{1, 2, \ldots, m\} \tag{3.3}$$

Zbiór zmiennych z_i definiuje zbiór IRI z $\mathbb{I}$ takich, że każdy z tych IRI jest obecny w wynikach przynajmniej dwóch z zapytań z $\mathbb{X}$:

$$z_i = \begin{cases} 1 & \exists j \exists k \ (j \neq k \wedge P_j, P_k \in \mathbb{X} \wedge I_i \in \mathbb{R}(P_j) \wedge I_i \in \mathbb{R}(P_k)) \\ 0 & \text{w przeciwnym przypadku} \end{cases} \qquad \forall i \in \{1, 2, \ldots, m\} \tag{3.4}$$

Z podanych definicji wynika, że $y_i - z_i = 1$ wtedy, i tylko wtedy, kiedy $\mathbb{X}$ zawiera dokładnie jeden wzorzec P_j taki, że $I_i \in \mathbb{R}(P_j)$, a zatem cel optymalizacji wyraża się jako

$$\sum_{i=1}^{m} (y_i - z_i) \tag{3.5}$$

Zgodnie z definicją $y_i = 1$ jeżeli I_i jest w wynikach dowolnego z wybranych wzorców, tzn. gdy $A_{i,j}x_j = 1$ dla dowolnego j. Można w tym celu przedstawić zbiór m nierówności:

$$My_i \geqslant \sum_{j=1}^{n} A_{i,j}x_j \qquad \forall i \in \{1, 2, \ldots, m\} \tag{3.6}$$

Żeby $y_i = 0$ wszystkie iloczny $A_{i,j}x_j$ muszą równać się 0, a zatem również ich suma:

$$y_i \leqslant \sum_{j=1}^{n} A_{i,j}x_j \qquad \forall i \in \{1, 2, \ldots, m\} \tag{3.7}$$

Żeby zdefiniować kiedy $z_i = 1$ używa się takich samych wyrażeń, ale pomniejszonych o 1:

$$Mz_i \geqslant -1 + \sum_{j=1}^{n} A_{i,j}x_j \qquad \forall i \in \{1, 2, \ldots, m\} \tag{3.8}$$

Do zapewnienia, że $z_i = 0$ używa się następujących nierówności:

$$M(1 - z_i) \geqslant 2 - \sum_{j=1}^{n} A_{i,j}x_j \qquad \forall i \in \{1, 2, \ldots, m\} \tag{3.9}$$

W celu zapewnienia, że co najmniej λ wzorców zostanie wybranych używa się nierówności:

$$\sum_{j=1}^{n} x_j \leqslant \lambda \tag{3.10}$$

Ostatecznie uzyskuje się model liniowy składający się z $4m + 1$ nierówności.

Rozszerzenie ontologii wymaga przetłumaczenia wzorców ze SPARQL na OWL 2 zgodnie z poniższymi krokami:

1. BGP i filtry są ekstrahowane ze wzorca.

2. BGP jest reprezentowany jako graf: każdy wzorzec trójki stanowi pojedynczy łuk, od wierzchołka odpowiadającemu podmiotowi do wierzchołka odpowiadającemu dopełnieniu i etykietowany predykatem.

3. Dla każdego z filtrów postaci `FILTER(zmienna op wartosc)`:

   a) dodaje się nowy wierzchołek dla `wartosc`;

   b) dodaje się łuk etykietowany operatorem `op` od wierzcholka odpowiadającemu zmiennej `zmienna` do nowo dodanego wierzchołka.

Ze względu na specyfikę Fr-ONT-Qu uzyskany graf jest drzewem ukorzenionym w wierzchołku odpowiadającemu zmiennej umieszczonej w głowie zapytania. Uzyskane drzewo jest następnie tłumaczone na wyrażenie klasowe OWL 2 EL przez funkcję przedstawioną w rozprawie jako Algorytm 3.1.

3.3 Wtyczka do środowiska *RapidMiner*

RapidMiner to narzędzie do wizualnego tworzenia przepływów pracy (ang. *workflow*) do celów eksploracji danych [17]. Pojedynczą jednostką wykonania w przepływie pracy jest *operator*, który z grubsza odpowiada funkcji w języku programowania. Każdy operator pobiera dane ze swoich portów wejściowych, przetwarza je i dostarcza do swoich portów wyjściowych. Połączenia pomiędzy portami operatorów w przepływie pracy definiują przepływ danych.

Podczas uczenictwa w projekcie *e-LICO: An e-Laboratory for Interdisciplinary Collaborative Research in Data Mining and Data-Intensive Science*[1] finansowanym z 7. Programu Ramowego Unii Europejskiej, zaprogramowane zostało *RMonto*, wtyczka rozszerzająca *RapidMinera* o możliwości przetwarzania danych Semantycznego Internetu [21]. Zarówno Fr-ONT-Qu jak i przedstawiony model matematyczny są dostępne jako operatory w *RMonto*.

[1] `http://e-lico.eu/`

Rozdział 4

Formalna Analiza Pojęć wspierana przez uczenie maszynowe

4.1 Formalna Analiza Pojęć

Formalna Analiza Pojęć (ang. *Formal Concept Analysis (FCA)*) to obszar matematyki stworzony, żeby sformalizować dyskurs o pojęciach i ich hierarchiach [29]. Kontekstem formalnym (ang. *formal context*) nazywa się trójkę (G, M, I) taką, że G jest zbiorem obiektów, M jest zbiorem binarnych atrybutów, a $I \subseteq G \times M$ to relacja binarna taka, że $(g, m) \in I$ wtedy, i tylko wtedy, kiedy g ma atrybut m. Operator derywacji (ang. *derivation operator*) $\cdot^I$ jest zdefiniowany dla dowolnego $X \subseteq G$ i dowolnego $Y \subseteq M$:

$$X^I = \{m \in M \colon \forall g \in X \colon (g, m) \in I\} \tag{4.1}$$

$$Y^I = \{g \in G \colon \forall m \in Y \colon (g, m) \in I\} \tag{4.2}$$

Pojęciem formalnym (ang. *formal concept*) nazywa się parę (A, B) taką, że A jest ekstensją (ang. *extension*), a B intensją (ang. *intension*), a ponadto zachodzą następujące równości: $A = B^I$ oraz $B = A^I$.

Na bazie pojęć formalnych definiuje się relację podpojęcia–nadpojęcia (ang. *subconcept–superconcept*) jako

$$(A_1, B_1) \prec (A_2, B_2) \iff A_1 \subset A_2 \iff B_1 \supset B_2 \tag{4.3}$$

Niech $\mathbb{K}$ będzie zbiorem wszystkich pojęć formalnych w kontekście formalnym (G, M, I). Relacja $\prec$ jest porządkiem częściowym, a zatem tworzy *kratę pojęć* (ang. *concept lattice*) kontekstu (G, M, I).

4.2 Obsługa niepełnej wiedzy

Niepełnym kontekstem formalnym (ang. *incomplete formal context*) nazywa się czwórkę $(G, M, \{+, ?, -\}, J)$ taką, że G jest zbiorem obiektów, M zbiorem atrybutów, a $J \subseteq G \times M \times \{+, ?, -\}$ jest relacją trójargumentową taką, że [8]:

- $(g, m, +) \in J$ jeżeli obiekt g ma atrybut m;
- $(g, m, -) \in J$ jeżeli obiekt g nie ma atrybutu m;
- $(g, m, ?) \in J$ jeżeli nie wiadomo czy obiekt g ma atrybut m.

Ponadto wymaga się, żeby dla każdej pary $(g, m) \in G \times M$ w J była dokładnie jedna z wyżej wymienionych trójek.

Kontekst formalny (G, M, I) nazywa się *uzupełnieniem* (ang. *completion*) niepełnego kontekstu formalnego $(G, M, \{+, ?, -\}, J)$ jeżeli spełnione są łącznie następujące warunki

- każda trójka $(g, m, +)$ implikuje, że $(g, m) \in I$;
- każda trójka $(g, m, -)$ implikuje, że $(g, m) \notin I$.

Częściowym opisem obiektu (ang. *partial object description*) jest trójka (g, A, S) taka, że:

- $g \in G$ jest obiektem w niepełnym kontekście formalnym;
- A jest zbiorem wszystkich atrybutów g;
- S jest zbiorem wszystkich atrybutów, których g nie ma.

Zbiór częściowych opisów obiektów $\{(g_1, A_1, S_1), \ldots, (g_n, A_n, S_n)\}$ odpowiada dokładnie niepełnemu kontekstowi formalnemu $(G, M, \{+, ?, -\}, J)$ takiemu, że:

- G składa się ze wszystkich obiektów w zbiorze: $G = \{g_1, \ldots, g_n\}$
- J zawiera trójki z + dla atrybutów w A_i, trójki z − dla atrybutów w S_i i trójki z ? dla pozostałych atrybutów.

Pełnym opisem obiektu (ang. *full object description*) nazywa się częściowy opis obiektu (g, A, S) taki, że $A \cup S = M$. Zbiór pełnych opisów obiektów dla wszystkich obiektów $g \in G$ odpowiada pojedynczemu kontekstowi formalnemu.

Częściowy opis obiektu (g, A, S) *jest rozszerzany przez* częściowy opis obiektu (g, A', S') jeżeli $A \subseteq A'$ oraz $S \subseteq S'$. Rozszerza się to pojęcie na niepełne konteksty formalne i przyjmuje, że niepełen kontekst formalny K_1 jest rozszerzany przez niepełen kontekst formalny K_2 jeżeli każdy częściowy opis obiektu w K_1 jest rozszerzany przez częściowy opis obiektu w K_2.

4.3 Implikacje pomiędzy atrybutami

Zbiór atrybutów X implikuje zbiór atrybutów Y, co oznacza się jako *implikację pomiędzy atrybutami* (ang. *attribute implication*) $X \rightarrow$Y, wtedy i tylko wtedy, gdy $X^I \subseteq Y^I$. Implikacja $L \rightarrow R$ jest *obalana* (ang. *refuted*) przez częściowy opis obiektu (g, A, S) jeżeli $L \subseteq A$, ale $R \cap S \neq \emptyset$. Implikacja jest obalana przez niepełen kontekst formalny jeżeli jest obalana przez którykolwiek z częściowych opisów obiektów w tym kontekście.

Niech $\mathcal{L}$ będzie zbiorem implikacji pomiędzy atrybutami i niech $P \subseteq M$ będzie zbiorem atrybutów. Domknięciem implikacyjnym (ang. *implicational closure*) P względem zbioru $\mathcal{L}$, oznaczanym przez $\mathcal{L}(P)$ nazywa się najmniejszy zbiór $Q \subseteq M$ taki, że:

- zawiera wszystkie atrybut P;
- dla każdej implikacji w $\mathcal{L}$, jeżeli przesłanka implikacji należy do Q, to należy również konkluzja.

Implikacja $L \rightarrow R$ *wynika* (ang. *follows*) ze zbioru implikacji $\mathcal{J}$ jeżeli $R \subseteq \mathcal{J}(L)$. Zbiór implikacji $\mathcal{J}$ jest *bazą implikacji* (ang. *implication base*) dla zbioru implikacji $\mathcal{L}$ jeżeli łącznie zachodzą poniższe warunki:

- poprawność (ang. *soundness*) – $\mathcal{L}$ zawiera wszystkie implikacje wynikające z $\mathcal{J}$;

- pełność (ang. *completeness*) – każda implikacja w $\mathcal{L}$ wynika z $\mathcal{J}$;
- minimalność (ang. *minimality*) – żaden ścisły podzbiór $\mathcal{J}$ nie ma jednocześnie obu powyższych własności.

Niech M będzie liniowo uporządkowany: $m_1 < m_2 < \ldots < m_n$. Dla dowolnych $A, B \subseteq M$, A *poprzedza* B *w porządku* i, oznaczane jako $A <_i B$, wtedy i tylko wtedy gdy:

- B zawiera m_i, podczas gdy A nie zawiera m_i;
- A oraz B oba zawierają dokładnie te same atrybuty poprzedzające m_i.

Porządkiem kratowym (ang. *lectic order*) na zbiorze M jest suma porządków $1, \ldots, n$.

4.4 Algorytm eksploracji atrybutów

Algorytm eksploracji atrybutów (ang. *attribute exploration algorithm*) to ogólna nazwa rodziny algorytmów, których celem jest uzyskanie pełnej wiedzy o implikacjach pomiędzy atrybutami w zadanym kontekście formalnym. Algorytm 4.1 w rozprawie przedstawia algorytm eksploracji atrybutów dedykowany dla ontologii OWL 2 EL [1]. Zakłada się, że istnieje pewien kontekst formalny $\overline{\mathcal{K}}$, znany użytkownikowi, ale nieznany algorytmowi. Algorytm ma dostęp wyłącznie do niepełnego kontekstu formalnego, którego uzupełnieniem jest $\overline{\mathcal{K}}$. Celem algorytmu jest obliczenie bazy implikacji zbioru wszystkich implikacji, które nie są obalane przez $\overline{\mathcal{K}}$ jednocześnie zadając tak mało pytań jak to możliwe.

4.5 Zastosowanie Formalnej Analizy Pojęć do uzupełniania ontologii

Za [1] przedstawiony jest sposób użycia algorytmu eksploracji atrybutów do uzupełniania ontologii bądź jej fragmentu. Niech $\mathcal{O}$ będzie spójną (ang. *consistent*) ontologią, a G zbiorem wszystkich nazwanych indywiduów (ang. *named individuals*). Niech M będzie dowolnym, skończonym zbiorem wyrażeń klasowych. Ontologia i oba zbiory łącznie tworzą niepełen kontekst formalny:

$$\mathcal{K} = \{(g, A, S) \colon g \in G \wedge A = \{C \in M \colon \mathcal{O} \models C(g)\} \wedge S = \{C \in M \colon \mathcal{O} \models \neg C(g)\}\} \tag{4.4}$$

Może on zostać wykorzystany jako wejście algorytmu eksploracji atrybutów. Każda odkryta implikacja między atrybutami $L \to R$ może zostać przedstawiona jako ogólne zawieranie się pojęć (ang. *general concept inclusion (GCI)*): $L_1 \sqcap \ldots \sqcap L_n \sqsubseteq R_1 \sqcap \ldots \sqcap R_m$, gdzie $L = \{L_1, \ldots, L_n\}$, a $R = \{R_1, \ldots, R_m\}$. W celu zapewnienia spójności między niepełnym kontekstem formalnym i ontologią, odkrywania baza implikacji i pojawiające się kontrprzykłady są dodawane na bieżąco do ontologii. Po wykonaniu algorytmu każdy GCI, który można utworzyć za pomocą przecięć i wyrażeń klasowych ze zbioru M albo wynika z ontologii albo jest sprzeczny z ontologią.

4.6 Zadanie klasyfikacji w Formalnej Analizie Pojęć

Główną wadą algorytmu eksploracji atrybutów jest przewaga eksploracji użytkownika nad eksploracją atrybutów: użytkownik musi odpowiedzieć na olbrzymią, potencjalnie wykładniczą, liczbę pytań. W [19, 24] zaproponowane został sposób na wykorzystanie Powiązanych Danych i uczenia maszynowego do pomocy użytkownikowi. Zamiast próbować całkowicie zastąpić użytkownika celem jest odpowiadanie na stawiane pytania jeżeli Powiązane Dane dostarczą dostatecznie dużo dowodów oraz przekierowywanie ich do użytkownika w przeciwnym razie. Wprowadza się stos ekspertów: najpierw jest narzędzie wnioskowania dedukcyjnego (ang. *reasoner*), później klasyfikator,

a na końcu użytkownik. Najpierw algorytm sprawdza za pomocą wnioskowania dedukcyjnego, czy implikacja nie wynika już z ontologii. Jeżeli nie, klasyfikator jest uruchamiany i dopiero jeżeli jego wiedza okaże się zbyt ograniczona, użytkownik musi udzielić odpowiedzi.

Formalny opis naszkicowanego algorytmu jest przedstawiony w rozprawie jako Algorytm 4.2. Algorytm wymaga ustawienia dwóch progów: θ_a oraz θ_r, będących odpowiednio, progami prawdopodobieństwa przyjęcia i odrzucenia implikacji bez pytania użytkownika.

Zazwyczaj funkcje rozpatrywane przez algorytmy klasyfikacji korzystają z wektorów liczbowych cech (ang. *features*), podczas gdy implikacja pomiędzy atrybutami to para zbiorów atrybutów. Żeby rozwiązać tę rozbieżność, zaproponowane zostało wykorzystanie zbioru miar używanych dla reguł asocjacyjnych (ang. *associataion rules*), ale obliczanych nie na bazie samej ontologii, ale raczej na bazie adekwatnego zbioru Powiązanych Danych. Żeby odpytywać zbiór, potrzebne jest odwzorowanie ze zbioru wyrażeń klasowych pełniących rolę atrybutów na wzorce SPARQL. Jako podstawowe rozwiązanie można zaproponować zapis atrybutów w składni Turtle.

Niech $\{L_1, \ldots, L_m\} \rightarrow \{R_i\}$ będzie implikacją między atrybutami, p liczbą obiektów, które mają wszystkie atrybuty $L_1, \ldots, L_m$, c liczbą obiektów z atrybutem R_i, a pc liczbą obiektów, które mają wszystkie atrybuty $L_1, \ldots, L_m, R_i$. Zgodnie z konwencją nazewniczą z [6], proponuje się wykorzystanie następujących miar jako cech implikacji:

coverage p

prevalence c

support pc

recall

$$\frac{pc}{c}$$

confidence

$$\frac{pc}{p}$$

Żeby obliczyć p, c oraz pc, należy zadać trzy zapytania do końcówki SPARQL. Te same miary mogą być obliczone przez użycie systemu wnioskowania dedukcyjnego na bazie ontologii. Ponadto, do zbioru cech warto dodać znormalizowaną długość przesłanki $\{L_1, \ldots, L_m\}$, tzn. $\frac{m}{|M|}$.

Należy również zauważyć, że zbiór przykładów uczących powstaje podczas pracy algorytmu i nie jest dostępny na początku procesu. W szczególności wymaga to wykorzystania algorytmów pozwalających na uczenie przyrostowe, np. zmodyfikowanej regresji logistycznej. Trzeba również zwrócić uwagę, że klasy mogą być silnie niezbalansowane i klasyfikator może faworyzować jedną z klas. Należy w związku z tym ważyć przykłady i stosować uczenie uwzględniające koszt (ang. *cost-sensitive learning*).

4.7 Implementacja

Przedstawiony algorytm został zaimplementowany w *Javie* i opublikowany w serwisie *GitHub*: `https://github.com/jpotoniec/FCA-ML`.

Rozdział 5

Swift Linked Data Miner

Swift Linked Data Miner (SLDM) to algorytm dedykowany odkrywaniu wyrażeń klasowych OWL 2 EL bezpośrednio z Powiązanych Danych i używając jedynie końcówki SPARQL jako sposobu na dostęp do danych. Odkryte wyrażenia są następnie wykorzystywane jako prawe strony aksjomatów zawierania się klas. SLDM został przedstawiony w [20], a jego implementacja w [23].

5.1 Pobieranie danych ze zdalnego grafu RDF

Niech $\mathcal{I}$ oznacza zbiór IRI. Należy rozpocząć od pobrania trójek RDF opisujących te IRI z końcówki SPARQL i zapewnienia odpowiedniej ich organizacji w pamięci. Ponieważ odkrywane są wyłącznie wyrażenia klasowe, wystarczy pobrać trójki, w których rozważane IRI występują jako podmioty.

Pobranie wszystkich trójek na raz może być niemożliwe. Żeby temu zaradzić, wprowadza się parametr *split* i dzieli zbiór $\mathcal{I}$ na rozłączne podzbiory zawierają po *split* elementów (z wyjątkiem, być może, ostatniego podzbioru). Dla każdego z tak uzyskanych podzbiorów tworzy się zapytanie SPARQL, które jest wysyłane do końcówki SPARQL w celu uzyskania rozłącznych zbiorów trójek, które ostatecznie są łączone w celu uzyskania takiego samego zbioru jak w wyniku zadania jednego zapytania bez podziału.

5.2 Strategie próbkowania

Jeżeli zbiór IRI jest duży, nawet takie podejście iteracyjne może okazać się zbyt wymagające czasowo i obliczeniowo. W takiej sytuacji należy dokonać próbkowania zbioru $\mathcal{I}$ za pomocą jednej z trzech proponowanych strategii. Strategia jednostajna (ang. *uniform strategy*) to proste próbkowanie losowe bez zwracania. Strategia zliczania predykatów (ang. *predicates counting strategy*) dokonuje bardziej ukierunkowanego wyboru przez używanie jako rozkładu prawdopodobieństwa znormalizowanej liczby różnych predykatów w trójkach RDF dla danego IRI. Strategia zliczania trójek (ang. *triples counting strategy*) działa bardzo podobnie, ale używa jako rozkładu prawdopodobieństwa liczby trójek zamiast liczby predykatów.

5.3 Organizacja danych w pamięci

Pobrane trójki muszą być ułożone w pamięci w sposób pozwalający na ich szybką i efektywną eksplorację. Przedstawiona organizacja nie wpływa na poprawność algorytmu, ale wpływa na jego efektywność.

Niech $\mathcal{T} = \{(s, p, o)\}$ będzie zbiorem trójek i niech P będzie zbiorem wszystkich predykatów występujących w trójkach. Pierwszy poziom indeksu to odwzorowanie z predykatów $p \in P$ na wskaźnik do drugiego poziomu. Niech O_p będzie zbiorem wszystkich obiektów występujących w trójkach, w których p jest predykatem. Na drugim poziomie, wskazywanym przez wskaźnik z pierwszego poziomu dla predykatu p, znajduje się odwzorowanie z elementów $o \in O_p$ na wektory na trzecim poziomie. Taki wektor składa się z podmiotów trójek, które mają predykat p i dopełnienie o.

Algorytm budowy indeksu na bazie zbioru trójek $\mathcal{T}$ jest przedstawiony w rozprawie jako Algorytm 6.1.

5.4 Podstawowe definicje

5.4.1 Wzorzec

Definicja 5.4.1 (wzorzec)**.** *Wzorzec* (ang. *pattern*) to dowolne wyrażenie klasowe albo zakres danych OWL 2 EL.

Żeby zdefiniować pojęcie pasowania indywiduum bądź literału do wzorca wprowadza się *funkcję dopasowującą* (ang. *matching function*) $\mu_{\mathbb{G}}$ taką, że $\mu_{\mathbb{G}}(a, C) = 1$ jeżeli a pasuje do C względem grafu $\mathbb{G}$ i $\mu_{\mathbb{G}}(a, C) = 0$ w przeciwnym wypadku. Funkcja dopasowująca jest zdefiniowana indukcyjnie.

Niech D będzie typem danych, a l literałem

$$\mu_{\mathbb{G}}(l, D) = \begin{cases} 1 & l^{LT} \in \text{ przestrzeni wartości } D \\ 0 & \text{w przeciwnym przypadku} \end{cases} \tag{5.1}$$

Niech m będzie literałem

$$\mu_{\mathbb{G}}(l, \{m\}) = \begin{cases} 1 & l^{LT} = m^{LT} \\ 0 & \text{w przeciwnym przypadku} \end{cases} \tag{5.2}$$

Niech D oraz E będą zakresami danych OWL 2 EL.

$$\mu_{\mathbb{G}}(l, D \text{ AND } E) = \mu(l, D) \cdot \mu(l, E) \tag{5.3}$$

Niech D będzie jednym z następujących typów danych: `owl:real`, `owl:rational`, `xsd:decimal`, `xsd:integer`, `xsd:dateTime`, `xsd:dateTimeStamp` i niech M będzie literałem typu D. Zakres danych $D[<= M]$ to ograniczenie przestrzeni wartości typu danych D do wartości nie większych niż M.

$$\mu_{\mathbb{G}}(l, D[<= M]) = \begin{cases} \mu_{\mathbb{G}}(l, D) & l^{LT} \leqslant M^{LT} \\ 0 & \text{w przeciwnym przypadku} \end{cases} \tag{5.4}$$

Niech D będzie jednym z następujących typów danych: `xsd:nonNegativeInteger`, `owl:real`, `owl:rational`, `xsd:decimal`, `xsd:integer`, `xsd:dateTime`, `xsd:dateTimeStamp`, tzn. jednym z typów danych wymienionych dla zakresu danych $D[<= M]$ bądź typem `xsd:nonNegativeInteger`. Niech m będzie literałem typu D. Zakres danych $D[>= m]$ to ograniczenie przestrzeni wartości typu danych D do wartości nie mniejszych niż m.

$$\mu_{\mathbb{G}}(l, D[>= m]) = \begin{cases} \mu_{\mathbb{G}}(l, D) & l \geqslant m \\ 0 & \text{w przeciwnym przypadku} \end{cases} \tag{5.5}$$

Niech C będzie klasą.

$$\mu_{\mathbb{G}}(a, C) = \begin{cases} 1 & (a, \texttt{rdf:type}, C) \in \mathbb{G} \\ 0 & \text{w przeciwnym przypadku} \end{cases} \tag{5.6}$$

Niech C oraz D będą wyrażeniami klasowymi.

$$\mu_{\mathbb{G}}(a, C \text{ AND } D) = \mu(a, C) \cdot \mu(a, D) \tag{5.7}$$

Niech a oraz b będą indywiduami.

$$\mu_{\mathbb{G}}(a, \{b\}) = \begin{cases} 1 & a = b \\ 1 & (a, \texttt{owl:sameAs}, b) \in \mathbb{G} \vee (b, \texttt{owl:sameAs}, a) \in \mathbb{G} \\ 0 & \text{w przeciwnym przypadku} \end{cases} \tag{5.8}$$

Niech p będzie własnością obiektową, a a oraz b indywiduami.

$$\mu_{\mathbb{G}}(a, p \text{ VALUE } b) = \begin{cases} 1 & (a, p, b) \in \mathbb{G} \\ 0 & \text{w przeciwnym przypadku} \end{cases} \tag{5.9}$$

Niech r będzie własnością danych, a indywiduum, a l literałem.

$$\mu_{\mathbb{G}}(a, r \text{ VALUE } l) = \begin{cases} 1 & (a, r, l) \in \mathbb{G} \\ 0 & \text{w przeciwnym przypadku} \end{cases} \tag{5.10}$$

Rozpatruje się ograniczenie zwrotne p SELF, gdzie p to własność obiektowa.

$$\mu_{\mathbb{G}}(a, p \text{ SELF}) = \begin{cases} 1 & (a, p, a) \in \mathbb{G} \\ 0 & \text{w przeciwnym przypadku} \end{cases} \tag{5.11}$$

Niech p będzie własnością obiektową, a C wzorcem.

$$\mu_{\mathbb{G}}(a, p \text{ SOME } C) = \begin{cases} 1 & \exists b\ [(a, p, b) \in \mathbb{G} \wedge \mu_{\mathbb{G}}(b, C) = 1] \\ 0 & \text{w przeciwnym przypadku} \end{cases} \tag{5.12}$$

Niech r będzie własnością danych, a D zakresem danych.

$$\mu_{\mathbb{G}}(a, r \text{ SOME } D) = \begin{cases} 1 & \exists l\ [(a, r, l) \in \mathbb{G} \wedge \mu_{\mathbb{G}}(l, D) = 1] \\ 0 & \text{w przeciwnym przypadku} \end{cases} \tag{5.13}$$

5.4.2 Częstotliwość wzorca

W celu zdefiniowania *częstego wzorca*, należy najpierw zdefiniować miarę częstości.

Definicja 5.4.2 (zbiór dowodowy)**.** Niech $\mathcal{I}$ będzie zbiorem *IRIs*, a $\mathcal{L}$ zbiorem literałów. *Zbiór dowodowy* (ang. *proof set*) S wzorca C to podzbiór zbioru $\mathcal{I} \cup \mathcal{L}$ taki, że jego elementy pasują do wzorca C:

$$S = \{s \in \mathcal{I} \cup \mathcal{L} \colon \mu_{\mathbb{G}}(s, C) = 1\} \tag{5.14}$$

Niech w będzie funkcją przypisującą elementom zbioru $\mathcal{I} \cup \mathcal{L}$ wagę z przedziału $[0, 1]$. Nazywa się taką funkcję *funkcją ważącą* (ang. *weighting function*). Jej celem jest przekazanie do algorytmu wiedzy o ważności danego indywiduum lub literału. Domyślna funkcja ważąca to $w^{def}(s) = \frac{1}{|\mathcal{I} \cup \mathcal{L}|}$ dla wszystkich $s \in \mathcal{I} \cup \mathcal{L}$, ale te wartości ulegają zmianie w miarę wykonania algorytmu.

Definicja 5.4.3 (wsparcie)**.** Niech C będzie wzorcem, a S jego zbiorem dowodowym. *Wsparcie* wzorca C względem funkcji ważącej w jest dane następującym wzorem:

$$\sigma(C, w) = \sum_{s \in S} w(s) = \sum_{s \in \mathcal{I} \cup \mathcal{L}} w(s) \mu_{\mathbb{G}}(s, C) \tag{5.15}$$

Definicja 5.4.4 (częsty wzorzec)**.** Niech dany będzie *próg minimalnego wsparcia* (ang. *minimal support threshold*) θ_σ. *Częstym wzorcem* (ang. *frequent pattern*) względem funkcji ważącej w nazywa się dowolny wzorzec C taki, że $\sigma(C, w) \geqslant \theta_\sigma$.

Definicja 5.4.5 (częsty predykat)**.** *Częsty predykat* to predykat p, który może być częścią częstego wzorca. W tym celu suma wag podmiotów obecnych w trójkach z predykatem p musi osiągnąć przynajmniej próg minimalnego wsparcia θ_σ. Zbiór dowodowy predykatu p jest zdefiniowany jako

$$S = \{s \in \mathcal{I} \cup \mathcal{L} \colon \exists o \, (s, p, o) \in \mathbb{G}\} \tag{5.16}$$

Wsparcie jest zdefiniowane spójnie ze wsparciem dla wzorca

$$\sigma(p, w) = \sum_{s \in S} w(s) \tag{5.17}$$

i wymaga się, żeby $\sigma(p, w) \geqslant \theta_\sigma$.

5.4.3 Długość wzorca

Długością wzorca C (ang. *pattern length*) $\mathfrak{l}(C)$ nazywa się powiększoną o 1 największą liczbę wzorców zagnieżdżonych we wzorcu C za pomocą kwantyfikacji egzystencjalnej.

5.5 Algorytm

5.5.1 Odkrywanie częstych typów danych

Konstrukcja algorytmu odkrywania częstych wzorców rozpoczyna się od rozważenia wzorców następującej postaci: typ danych D oraz ograniczenia przestrzeni wartości typu danych D. W celu odkrycia takich wzorców, zaproponowana została funkcja przedstawiona w rozprawie jako Algorytm 6.2.

5.5.2 Odkrywanie częstych wzorców z ustalonym dopełnieniem

Algorytm 6.3 w rozprawie przedstawia funkcję do odkrywania wzorców składających się wyłącznie z pojedynczej klasy A. Ten pomysł może być rozszerzony na dowolny predykat p, w celu utworzenia wzorców postaci $p \,\textsc{value}\, a$ oraz $r \,\textsc{value}\, l$, gdzie a to indywiduum, a l to literał. Formalny algorytm został przedstawiony w rozprawie jako Algorytm 6.4. Ten sam pomysł może być też wykorzystany do odkrywania ograniczeń zwrotnych $p \,\textsc{self}$ i został przedstawiony jako w rozprawie Algorytm 6.5. W końcu można rozpatrzyć wyliczenie indywiduów postaci $\{a\}$ oraz wyliczenie literałów $\{l\}$. Obecność takich wzorców jest określona wyłącznie przez funkcję ważącą w i nie zależy od trzypoziomowego indeksu. Funkcja korzystająca z tego spostrzeżenia jest przedstawiona w rozprawie jako Algorytm 6.6.

5.5.3 Odkrywanie przecięć

Po odkryciu zbioru wzorców praktycznym jest połączenie ich w dłuższe wzorce używając operatora AND. Pomaga to w prezentacji odkrytych wzorców w bardziej skondensowanej formie i

pozwala na większą ekspresywność. W tym celu SLDM używa Algorytmu 6.7 z rozprawy, żeby połączyć zbiór wzorców odkrytych nad danym zbiorem IRI w zamknięte koniunkcje (ang. *closed intersections*), czyli koniunkcje, których zbiory dowodowe i wsparcia każdego z argumentów są identyczne.

5.5.4 Algorytm odkrywania wzorców

Zarys algorytmu *Swift Linked Data Miner* (SLDM) jest przedstawiony jako funkcja `SLDM` jako w Algorytmie 5.1 (Algorytm 6.8 w rozprawie). W celu oszczędzania pamięci, uzyskany indeks jest oczyszczany względem funkcji ważącej w, używając funkcji przedstawionej jako Algorytm 6.9 w rozprawie, która usuwa części indeksu odpowiadające predykatom, które nie są częste. Gdy trzypoziomowy indeks jest oczyszczony, algorytm iteruje po wszystkich predykatach, które w nim pozostały i tworzy tymczasowe zbiory wzorców $\mathcal{P}_p$.

Jeżeli taki zbiór okaże się pusty, funkcja `MineSome` przedstawiona w rozprawie jako Algorytm 6.10 jest używana do odkrywania wzorców postaci $p\,\text{SOME}\ldots$. W tym celu jest obliczana nowa funkcja ważąca w_{new}, która przypisuje wagę każdemu dopełnieniu o obecnemu na drugim poziomie trzypoziomowego indeksu dla predykatu p.

Używając uzyskanych tymczasowych struktur, SLDM jest wywoływany rekurencyjnie ze zwiększoną długością wzorca $d+1$, a uzyskane wzorce wraz z ich zbiorami dowodowymi, są przechowywane w zbiorze $\mathcal{P}$. Gdy wywołanie rekurencyjne się zakończy, odkryte wzorce w $\mathcal{P}$ są dopasowywane przez dodanie prefiksu $p\,\text{SOME}$ i skonstruowanie odpowiadających im zbiorów dowodowych na bazie zbiorów dowodowych z wykonania rekurencyjnego.

```
1  function SLDM(𝓘, 𝓛, w, d)
2      𝒫 ←MineDatatype(𝓛, w)∪MineEnum(𝓘, 𝓛, w)
3      𝒯 ← pobrane trójki mające podmioty w zbiorze 𝓘, jak przedstawiony w Section 5.3
4      ℑ ←BuildIndex(𝒯)
5      ℑ ←PruneIndex(ℑ, w)
6      foreach p ∈ ℑ do
7          if p == rdf:type then
8              𝒫_p ← MineType(ℑ, w)
9          end
10         else
11             𝒫_p ← MineValue(ℑ, w, p) ∪ MineSelf(ℑ, w, p)
12         end
13         if 𝒫_p == ∅ ∧ d < θ_t then
14             𝒫_p ← MineSome(ℑ, w, p, d)
15         end
16         𝒫 ← 𝒫 ∪ 𝒫_p
17     end
18     return MineClosedIntersections(𝒫)
19 end
```

Algorithm 5.1: Funkcja, która łączy pobieranie, indeksowanie i rekurencyjne odkrywanie w Swift Linked Data Miner. Jej argumentami są zbiór IRI $\mathcal{I}$, zbiór literałów $\mathcal{L}$, funkcja ważąca w i aktualna długość wzorca d.

5.6 Wtyczka do *Protégé*

Swift Linked Data Miner został zaimplementowany w języku *Java*. Jest otwartoźródłowy i dostępny w repozytorium *Git*[1] `http://bitbucket.com/jpotoniec/sldm`. Taki wybór technologi umożliwił integrację SLDM jako wtyczki do *Protégé*, popularnego środowiska do edycji ontologii [14]. SLDM został również integrowany z *WebProtégé*, prostszą wersją *Protégé*, działającą wyłącznie w przeglądarce internetowej [13]. Integracja z *WebProtégé* została wykonana w ramach pracy inżynierskiej [27] i później zaprezentowana na konferencji [28].

5.7 Ocena eksperymentalna

W celu ewaluacji algorytmu, zdecydowano o użyciu *DBpedii* jako grafu RDF i przetłumaczenie odkrytych aksjomatów na angielski. Użyto DBpedii 2015-04 i rozpoczęto od wykonania eksploracyjnej analizy danych w celu zebrania podstawowych statystyk o grafie. Analizując uzyskane wyniki, wybrano 5 klas wraz z ich rodzicami i dziadkami, i uzyskano taksonomię 14 klas.

Dla każdej z tych klas uruchomiono SLDM dwa razy: pierwsze uruchomienie odbyło się dla progu minimalnego wsparcia $\theta_\sigma = 0{,}5$, a drugie dla progu minimalnego wsparcia $\theta_\sigma = 0{,}8$. W ten sposób uzyskano 14 zbiorów aksjomatów dla każdego z dwóch wykonań SLDM. Uzyskane aksjomaty zostały przetłumaczone na angielski używając narzędzia opartego na *Attempto Controlled English*, dobre zdefiniowanym wariancie angielskiego odpowiednim do wyrażania formalnej wiedzy bez utraty precyzji [26].

Ze zbioru aksjomatów odkrytych z progiem $\theta_\sigma = 0{,}8$ uzyskano 56 pytań w złotym standardzie (ang. *gold standard*) i 168 normalnych pytań, podczas gdy dla drugiego zbioru uzyskano, odpowiednio, 61 i 425 pytań. Pytania zostały umieszczone w serwisie *crowdsourcingowym CrowdFlower*[2] i wymagano, aby na każde normalne pytanie odpowiedziało co najmniej 20 wykonawców. Każdy z wykonawców musiał odpowiadać również na pytania ze złotego standardu i wymagano, żeby odpowiedział poprawnie na przynajmniej 70% z zadanych pytań.

Zlecenie w serwisie *CrowdFlower* zostało zakończone w ciągu kilku godzin i kosztowało około 35 dolarów amerykańskich. Do każdego z pytań przypisano trzy liczby: liczbę wykonawców, którzy odpowiedzieli, odpowiednio, *Tak*, *Nie* oraz *Nie wiem* na zadanie pytanie. Zagregowane wyniki zostały przedstawione w Tablicy 5.1 i pokazują, że większość z odkrytych aksjomatów została oznaczona jako poprawne przez większość wykonawców, tzn., że byłyby to wartościowe propozycje dla inżyniera ontologii.

5.8 Charakterystyka obliczeniowa SLDM

5.8.1 Stabilność uzyskanych wyników

W celu określenia jak dobrze SLDM zachowuje się w zależności od początkowego zbioru IRI, przeprowadzony został następujący eksperyment. Ustalono rozmiar próbki na 250 i próg minimalnego wsparcia θ_σ na 0,95. Używając `random.org` wybrano dziesięć różnych ziaren generatora losowego (ang. *random seeds*) i uruchomiono SLDM używając każdego z tych ziaren dla każdej z pięciu klas z dołu hierarchii wybranych klas. W każdym przypadku co najmniej 68% aksjomatów logicznie wynikało z każdego ze zbiorów, a co najmniej 87,5% z co najmniej połowy zbiorów.

[1] `https://git-scm.com/`
[2] `https://www.crowdflower.com/`

Tablica 5.1: Podsumowanie wyników uzyskanych w eksperymencie *crowdsourcingowym*. Każde z pytań zostało zadane 20 wykonawcom, a następnie przypisane do kategorii w zależności od liczby odpowiedzi każdego typu.

Tak	*Nie*	*Nie wiem*	$\theta_\sigma = 0.5$		$\theta_\sigma = 0.8$	
16–20	0–5	0–5	295	69.41%	65	38.69%
11–15	6–10	0–5	40	9.41%	41	24.40%
11–15	0–5	0–5	78	18.35%	45	26.79%
6–10	11–15	0–5	3	0.71%	3	1.79%
6–10	6–10	0–5	5	1.18%	13	7.74%
0–5	16–20	0–5	2	0.47%	0	0.00%
0–5	11–15	0–5	2	0.47%	1	0.60%
łącznie			425	100.00%	168	100.00%

5.8.2 Analiza strategii próbkowania

W celu określenia jak różne strategie próbkowania wpływają na uzyskiwane wyniki, obliczono 450 zbiorów aksjomatów używając różnych ustawień parametrów. W 99/150, czyli 66% przypadków, strategia zliczania predykatów dała największą (włączając remisy) liczbę odkrytych aksjomatów.

W celu przeprowadzenia dalszych porównań, w każdym ze 150 zbiorów różnych ustawień parametrów z wyjątkiem strategi, postawiliśmy następujące pytanie: czy któryś z trzech zbiorów aksjomatów w pełni zawiera logicznie którykolwiek z pozostałych. Aksjomaty odkryte używając obu strategi zliczających zawierają aksjomaty odkryte ze strategią jednostajną w 81% przypadków, podczas gdy zawieranie w przeciwną stronę zachodzi w mniej niż 50% przypadków. To wskazuje, że strategie zliczające generują bardziej ogólne wyniki niż strategia jednostajna.

5.8.3 Wydajność obliczeniowa SLDM

W celu zbadania wydajności obliczeniowej SLDM, wybrano klasę `dbo:Book` i wykonano następujący eksperyment. Utworzono 300 kombinacji parametrów i dla każdej wykonano 10 uruchomień SLDM, żeby uwzględnić naturalną zmienność systemu komputerowego. W trakcie każdego z wykonań mierzono łączny czas procesora zużyty przez SLDM, liczbę zapytań SPARQL zadanych do końcówki SPARQL oraz zużycie pamięci. Czas procesora nigdy nie przekroczył 120 sekund, a liczba zapytań nigdy nie przekroczyła 400. Maksymalna potrzebna ilość pamięci była mniejsza niż 10GB.

Rozdział 6

Podsumowanie

Głównym wkładem rozprawy jest opracowanie zbioru metod automatycznego rozszerzania ontologii z Powiązanych Danych. Przedstawiono trzy metody uczenia się ontologii z Powiązanych Danych, każda podchodząca do problemu z innej perspektywy i rozwiązująca inny specyficzny problem.

Pierwsza z metod używa modelowania matematycznego w celu dokonania optymalnego wyboru podzbioru wzorców dostarczonych przez generator wzorców. Użycie modelu matematycznego dostarcza formalnej definicji problemu jako zadania optymalizacji. Metoda używa ogólnego solwera w celu dostarczenia rozwiązania typu *anytime*: im dłużej solwer jest uruchomiony, tym lepsze końcowe rozwiązanie jest uzyskiwane. Jeżeli solwer zostanie zakończony zanim obliczenia zostaną zakończone, dostarczy on heurystycznego rozwiązania problemu. Metoda generuje wyrażenia klasowe w OWL 2 EL, które mogą zostać nazwane i użyte jako nowe, zdefiniowane klasy w ontologii. W związku z tym nadaje się ona do rozszerzania istniejącej hierarchii klas. Zaproponowana metoda została zaimplementowana w *RMonto*, wtyczce do środowiska do eksploracji danych *RapidMiner*. Jest ona oparta na wcześniejszych pracach [22], a szczegóły opisane są w Rozdziale 4 rozprawy.

Druga metoda pozwala na uzupełnianie hierarchii klas ontologii o brakujące subsumpcje. Podejście jest oparte na Formalnej Analizie Pojęć (FCA), dobrze zdefiniowanym systemie do analizy zależność w kratach pojęć. Integruje ono algorytm eksploracji atrybutów z klasyfikatorem, w celu wsparcia użytkownika przez odpowiadanie na niektóre z pytań stawianych przez algorytm. Cechy używane przez klasyfikator są obliczane na bazie Powiązanych Danych, a odpowiedzi udzielane przez użytkownika są używane jako etykiety w zadaniu klasyfikacji. Podejście jest odpowiednie do rozszerzania istniejących ontologii o dodatkowe aksjomaty zawierania się klas w obrębie istniejącej hierarchii klas. Przedstawiona metoda została zaimplementowana jako samodzielna aplikacja w języku *Java*. Metoda została zaprezentowana w [19, 24] i opisana w Rozdziale 5 rozprawy.

Trzecie podejście, nazwane Swift Linked Data Miner (SLDM), to algorytm odkrywania częstych wzorców, w którym przestrzeń wzorców pokrywa wszystkie wyrażenia klasowe dopuszczalne w OWL 2 EL jako nadklasy w aksjomatach zawierania się klas. Algorytm nie stosuje typowej strategi *utwórz i przetestuj*, w której wzorce są najpierw tworzone, a następnie sprawdzane jest czy są one częste czy nie. Zamiast tego SLDM opiera się na sprytnej organizacji trójek RDF w pamięci, co umożliwia efektywne odkrywanie częstych wzorców. SLDM jest połączony z podejściem do iteracyjnego pobierania adekwatnych części ze zbioru Powiązanych Danych w trakcie odkrywania wzorców, biorącym pod uwagę konieczność optymalizacji liczby i złożoności zadawanych zapytań SPARQL, a także ze strategiami oszczędzania zasobów za pomocą próbkowania. Pokazano, że SLDM jest dobrym wyborem jeżeli chodzi o dodawanie nowych aksjomatów do ontologii opisujących już istniejące w ontologii klasy. SLDM został zaimplementowany w języku *Java* i istnieje

możliwość wykorzystania go jako bibliotekę bądź bezpośrednio w środowiskach do edycji ontologii *Protégé* i *WebProtégé*. SLDM został przedstawiony w [20, 23] i jest opisany w Rozdziale 5.

Z trzech przedstawionych metod, SLDM zdaje się być najbardziej przyjazny dla użytkownika. Metoda oparta na modelowaniu matematycznym generuje wyrażenia klasowe, które są równoważne nowym klasom, ale nie jest w stanie wygenerować ich opisów i nazw. Użytkownik jest zmuszony do analizy uzyskanych wyników i podjęcia decyzji na temat ich semantyki w języku naturalnym. Algorytm eksploracji atrybutów wymaga dużo uwagi od użytkownika nawet kiedy jest połączony z klasyfikatorem. Algorytm zmusza użytkownika do odpowiadania na pytania, z których niektóre są trudne do zrozumienia, a proces jest męczący i podatny na błędy. Dla porównania, SLDM jest prawie bezobsługowy: w najprostszym ujęciu użytkownik podaje wyłącznie klasę, którą jest zainteresowany i adres końcówki SPARQL, a następnie przyjmuje bądź odrzuca zaproponowane aksjomaty.

Porównując potrzebny czas obliczeń, SLDM również okazuje się najlepszy. W przypadku pierwszej metody, zero-jedynkowe programowanie matematyczne jest problemem NP-zupełnym i rozwiązanie go może zająć dużo czasu, spowalniając cały proces. Celem wyposażenia algorytmu eksploracji atrybutów w klasyfikator było skrócenie czasu niezbędnego do zakończenia całego procesu. Chociaż cel został osiągnięty, nadal potrzeba dużo czasu, ponieważ klasyfikator musi najpierw zostać wytrenowany i dopiero wtedy może odpowiadać na niektóre z pytań zadawanych przez algorytm. SLDM jest zoptymalizowany ze względu na potrzebny czas przez użycie trzypoziomowego indeksu, dzielenie zapytań zadawany do końcówki SPARQL i próbkowanie.

W porównaniu z istniejącymi wcześniej metodami, główną zaletą zaproponowanych metod jest możliwość pracy wyłącznie z grafem RDF dostępnym w końcówce SPARQL. Nie ma potrzeby pobierania całego grafu i przetwarzania go offline. Ponadto, żadna z przedstawionych metod nie wymaga wiedzy dziedzinowej o tym co jest typowe bądź popularne w modelowaniu ontologii, jak praca Bühmanna i Lehmanna [7]. Umożliwia to łatwiejszą adaptację do wiecznie zmieniającego się krajobrazu Powiązanych Danych. W końcu, SLDM może być uruchomiony w trybie, który oszczędza zasoby końcówki SPARQL, przez ograniczenie złożoności i liczby zadawanych zapytań. Jest to nowa cecha, jako że proponowane wcześniej metody zwykle używały złożonych zapytań SPARQL 1.1 z klauzulami COUNT i GROUP BY.

Przedstawione metody mogą znaleźć różne zastosowania. Na przykład, graf RDF może być skonstruowany przez społeczność, co wprowadza dużą ilość szumu i niezgodności, jak można zaobserwować to w *DBpedii*. Użyteczną byłaby ekstrakcja i formalizacja wiedzy, a następnie użycie jej do ukierunkowania dalszego rozwoju grafu. Inne zastosowanie jest związane z ekstrakcją wiedzy z nieustrukturyzowanego tekstu: algorytm tworzy graf RDF bez żadnej ontologii na bazie danego tekstu, następnie ontologia jest automatycznie odkrywana i używana do kierowania dalszą ekstrakcją. W ten sposób graf jest odkrywany z danych, ale jednocześnie wyposażony w wiedzę taksonomiczną i spójne użycie słownictwa.

Każda z metod może zostać rozszerzona i aktualnie proponuje się następujące kierunki rozwoju. Metoda oparta na modelowaniu matematycznym wymaga najpierw materializacji wzorców znalezionych w danych i dopiero wtedy model może zostać wygenerowany i rozwiązany. Interesującym pytaniem jest jak uniknąć materializacji i generować model bezpośrednio z grafu RDF. Wygenerowane wyrażenia klasowe są pozbawione opisów i w związku z tym mogą być trudne do zrozumienia dla użytkownika. Dostarczenie tekstowego opisu, który byłby powiązany z opisami słownictwa użytego w wyrażeniu klasowym, ale jednocześnie nie byłby sztucznym sklejeniem przy wykorzystaniu kontrolowanego języka, jest z pewnością interesującym wyzwaniem. W końcu, analiza i porównanie różnych możliwych sformułowani celu optymalizacji może być interesujące. Być może inżyniera ontologii nie jest problemem typu "jeden rozmiar pasuje na każdego"i różne cechy

muszą być brane pod uwagę w różnych kontekstach.

Główną wadą drugiej metody jest czas niezbędny do wytrenowania klasyfikatora. Zastąpienie jednego klasyfikatora innym, wymagającym mniejszej liczby przykładów, spowodowałoby tylko zmniejszenie problemu, a nie usunięcie go. Rozwiązanie problemu wymaga uczenia z przeniesieniem (ang. *transfer learning*), tzn. jakiegoś rodzaju transferu wiedzy uzyskanej w jednym wykonaniu na kolejne wykonania. Wymaga to cech do klasyfikacji, które są niezależne od ontologii albo tłumaczenia cech pomiędzy ontologiami. Prawdopodobnie najlepszym scenariuszem byłoby opracowanie cech, które są zarówno niezależne od ontologii, jak i od użytkownika, i dostarczanie użytkownikowi wstępnie wytrenowanego klasyfikatora, który jest tylko dostosowywany w trakcie wykonania.

Oczywistym ograniczeniem Swift Linked Data Miner jest przestrzeń wzorców pokrywająca wyłącznie język OWL 2 EL. Chociaż jest to istotny profil OWL 2, rozszerzenie przestrzeni, żeby pokryła całą specyfikację OWL 2 pozwoliłoby użytkownikowi na dokonanie wyboru konstruktorów, które mają być używane w odkrywanych aksjomatach, a więc ich ekspresyjności i złożoności procesu wnioskowania.

Podziękowania

Autor dziękuje za wsparcie Narodowemu Centrum Nauki (Grant nr 2013/11/N/ST6/03065), programowi POMOST Fundacji na rzecz Nauki Polskiej (Grant No POMOST/2013-7/8) współfinansowanemu ze środków Programu Rozwoju Regionalnego Unii Europejskiej oraz z grantu 09/91/DSPB/0627.

Literatura

[1] Franz Baader, Bernhard Ganter, Baris Sertkaya, Ulrike Sattler. Completing description logic knowledge bases using formal concept analysis. Manuela M. Veloso, redaktor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, strony 230–235, 2007.

[2] Franz Baader, Carsten Lutz, Sebastian Brandt. Pushing the EL envelope further. Kendall Clark, Peter F. Patel-Schneider, redaktorzy, *Proceedings of the Fourth OWLED Workshop on OWL: Experiences and Directions, Washington, DC, USA, 1-2 April 2008.*, wolumen 496 serii *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[3] Tim Berners-Lee. Linked data. Pobrane w dniu 2016-09-26 z `https://www.w3.org/DesignIssues/LinkedData.html`.

[4] Tim Berners-Lee, James Hendler, Ora Lassila. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[5] Christian Bizer, Tom Heath, Tim Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[6] Yannick Le Bras, Philippe Lenca, Stéphane Lallich. Optimonotone measures for optimal rule discovery. *Computational Intelligence*, 28(4):475–504, 2012.

[7] Lorenz Bühmann, Jens Lehmann. Pattern based knowledge base enrichment. Harith Alani, Lalana Kagal, Achille Fokoue, Paul T. Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha F. Noy, Chris Welty, Krzysztof Janowicz, redaktorzy, *The Semantic Web - ISWC 2013*, wolumen 8218 serii *Lecture Notes in Computer Science*, strony 33–48. Springer, 2013.

[8] Peter Burmeister, Richard Holzer. Treating incomplete knowledge in formal concept analysis. Bernhard Ganter, Gerd Stumme, Rudolf Wille, redaktorzy, *Formal Concept Analysis: Foundations and Applications*, strony 114–126. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[9] Gavin Carothers, Eric Prud'hommeaux. RDF 1.1 turtle. W3C recommendation, W3C, Luty 2014. `http://www.w3.org/TR/2014/REC-turtle-20140225/`.

[10] M. Duerst, M. Suignard. Internationalized resource identifiers (iris). RFC 3987, RFC Editor, January 2005. `http://www.rfc-editor.org/rfc/rfc3987.txt`.

[11] Bernardo Cuenca Grau, Peter Patel-Schneider, Boris Motik. OWL 2 web ontology language direct semantics (second edition). W3C recommendation, W3C, Grudzie/n 2012. `http://www.w3.org/TR/2012/REC-owl2-direct-semantics-20121211/`.

[12] Steven Harris, Andy Seaborne. SPARQL 1.1 query language. W3C recommendation, W3C, Marzec 2013. `http://www.w3.org/TR/2013/REC-sparql11-query-20130321/`.

[13] Matthew Horridge, Tania Tudorache, Csongor Nyulas, Jennifer Vendetti, Natalya Fridman Noy, Mark A. Musen. Webprotégé: a collaborative web-based platform for editing biomedical ontologies. *Bioinformatics*, 30(16):2384–2385, 2014.

[14] Holger Knublauch, Ray W. Fergerson, Natalya Fridman Noy, Mark A. Musen. The Protégé OWL plugin: An open development environment for semantic web applications. Sheila A. McIlraith, Dimitris Plexousakis, Frank van Harmelen, redaktorzy, *The Semantic Web - ISWC 2004: Third International Semantic Web Conference,Hiroshima, Japan, November 7-11, 2004. Proceedings*, wolumen 3298 serii *Lecture Notes in Computer Science*, strony 229–243. Springer, 2004.

[15] Agnieszka Lawrynowicz, Jedrzej Potoniec. Pattern based feature construction in semantic data mining. *Int. J. Semantic Web Inf. Syst.*, 10(1):27–65, 2014.

[16] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, Christian Bizer. DBpedia - A large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[17] Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, Timm Euler. YALE: rapid prototyping for complex data mining tasks. Tina Eliassi-Rad, Lyle H. Ungar, Mark Craven, Dimitrios Gunopulos, redaktorzy, *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*, strony 935–940. ACM, 2006.

[18] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Achille Fokoue, Zhe Wu. OWL 2 web ontology language profiles (second edition). W3C recommendation, W3C, Grudzie/n 2012. `http://www.w3.org/TR/2012/REC-owl2-profiles-20121211/`.

[19] Jedrzej Potoniec. Towards ontology refinement by combination of machine learning and attribute exploration. Patrick Lambrix, Eero Hyvönen, Eva Blomqvist, Valentina Presutti, Guilin Qi, Uli Sattler, Ying Ding, Chiara Ghidini, redaktorzy, *Knowledge Engineering and Knowledge Management - EKAW 2014 Satellite Events, VISUAL, EKM1, and ARCOE-Logic, Linköping, Sweden, November 24-28, 2014. Revised Selected Papers.*, wolumen 8982 serii *Lecture Notes in Computer Science*, strony 225–232. Springer, 2014.

[20] Jedrzej Potoniec, Piotr Jakubowski, Agnieszka Ławrynowicz. Swift Linked Data Miner: Mining OWL 2 EL class expressions directly from online RDF datasets . *Joural of Web Semantics*. DOI: 10.1016/j.websem.2017.08.001.

[21] Jedrzej Potoniec, Agnieszka Lawrynowicz. RMonto: Ontological extension to RapidMiner. *Poster and Demo Session of the ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany*, 2011.

[22] Jedrzej Potoniec, Agnieszka Lawrynowicz. Combining ontology class expression generation with mathematical modeling for ontology learning. Blai Bonet, Sven Koenig, redaktorzy, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, strony 4198–4199. AAAI Press, 2015.

[23] Jedrzej Potoniec, Agnieszka Lawrynowicz. A Protégé plugin with Swift Linked Data Miner. Takahiro Kawamura, Heiko Paulheim, redaktorzy, *Proceedings of the ISWC 2016 Posters & Demonstrations Track co-located with 15th International Semantic Web Conference (ISWC 2016), Kobe, Japan, October 19, 2016.*, wolumen 1690 serii *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.

[24] Jedrzej Potoniec, Sebastian Rudolph, Agnieszka Lawrynowicz. Towards combining machine learning with attribute exploration for ontology refinement. Matthew Horridge, Marco Rospocher, Jacco van Ossenbruggen, redaktorzy, *Proceedings of the ISWC 2014 Posters & Demonstrations Track a track within the 13th International Semantic Web Conference, ISWC 2014, Riva del Garda, Italy, October 21, 2014.*, wolumen 1272 serii *CEUR Workshop Proceedings*, strony 229–232. CEUR-WS.org, 2014.

[25] Max Schmachtenberg, Christian Bizer, Anja Jentzsch, Richard Cyganiak. Linking open data cloud diagram 2014. Rozpowszechniane na licencji CC-BY-SA, pobrane w dniu 2016-09-26 z `http://lod-cloud.net/`.

[26] Rolf Schwitter, Norbert E. Fuchs. Attempto controlled english (ACE) A seemingly informal bridgehead in formal territory (poster abstract). Michael J. Maher, redaktor, *Logic Programing, Proceedings of the 1996 Joint International Conference and Syposium on Logic Programming, Bonn, Germany, September 2-6, 1996*, strona 536. MIT Press, 1996.

[27] Tomasz Sosnowski. Rozszerzenie WebProtégé o możliwość indukcji aksjomatów z powiazanych danych. Praca inżynierska, Politechnika Poznańska 2016.

[28] Tomasz Sosnowski, Jedrzej Potoniec, Agnieszka Ławrynowicz. Swift linked data minerextension for webprotégé. P. Ciancarini, F. Poggi, M. Horridge, J. Zhao, T. Groza, M. C. Suárez-Figueroa, M. d'Aquin, V. Presutti, redaktorzy, *Knowledge Engineering and Knowledge Management - EKAW 2016 Satellite Events, EKM and Drift-a-LOD. Bologna, Italy, November 19–23, 2016. Revised Selected Papers*, wolumen 10180 serii *Lecture Notes in Computer Science*. Springer, 2017.

[29] Rudolf Wille. Formal concept analysis as mathematical theory of concepts and concept hierarchies. Bernhard Ganter, Gerd Stumme, Rudolf Wille, redaktorzy, *Formal Concept Analysis: Foundations and Applications*, strony 1–33. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[30] David Wood, Markus Lanthaler, Richard Cyganiak. RDF 1.1 concepts and abstract syntax. W3C recommendation, W3C, Luty 2014. `http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/`.