K-Tree(n) - czyli efektywne grupowanie kolekcji dokumentów XML

Anna Leśniewska[1], **Kamila Rzeźnik**[1]

Streszczenie: Artykuł przedstawia propozycję metody grupowania dokumentów XML, w oparciu o reprezentację dokumentów XML w postaci uporządkowanego, etykietowanego drzewa. Zaproponowana metoda porządkuje kolekcję dokumentów XML, uwzględniając wyłącznie strukturę dokumentów i pomijając zawartość tekstową. Zaproponowano wykorzystanie miary podobieństwa dwóch drzew jako stosunek wielkości największego poddrzewa wspólnego dla obu drzew, do wielkości większego z obu drzew. Zaproponowano algorytm heurystyczny (K-Tree(n)), do znajdowania maksymalnego wspólnego poddrzewa. Do zadania grupowania wybrano metodę AHC (Agglomerative Hierarchical Clustering). W metodzie AHC odległości pomiędzy dokumentami zastąpiono podobieństwem, wyznaczonym przez heurystykę.

Słowa kluczowe: eksploracja danych, grupowanie xml

1. Wprowadzenie

eXtensible Markup Language, znany szerzej jako XML, stał się standardem w rozwijaniu wielu aplikacji, w szczególności aplikacji Webowych. Wzrost popularności XML spowodował potrzebę opracowania nowych technik umożliwiających przetwarzanie tego rodzaju danych. Znalezienie odpowiedniego mechanizmu do odkrywania wiedzy poprzez metody eksploracji danych np. klastrowanie (grupowanie) dokumentów XML, staje się kluczowym dla eksploracji Web lub innych systemów, które taki rodzaj dokumentów przechowują oraz umożliwiają przetwarzanie, np. biblioteki cyfrowe. W dokumentach XML, oprócz zawartości, doszedł jeszcze jeden nośnik informacji, ich struktura. Fakt, że informacja w dokumentach XML zawarta jest nie tylko wewnątrz znaczników i atrybutów, ale również w strukturze dokumentu, powoduje, że dokumenty XML nie można poddać klasycznym metodom eksploracji danych. W eksploracji danych semistrukturalnych należy zastosować metody uwzględniające ich złożoną budowę, umożliwiającą eksploracje zawartości przechowywanych w semistrukturach i/lub ich struktury. Poniższa praca przedstawia opracowanie metody grupowania dokumentów XML biorąc pod uwagę ich strukturalne podobieństwo. Jednym z możliwych zastosowań grupowania dokumentów XML ze względu na ich strukturę może być organizowanie kolekcji XML w mniejsze zbiory o charakterystycznej strukturze, co umożliwia wspomaganie optymalizacji tworzenia zapytań.

[1] Wydział Informatyki i Zarządzania, Politechnika Poznańska, Piotrowo 2, 60-965 Poznań
e-mail: {alesniewska}@cs.put.poznan.pl

1.1. Cel pracy

Celem pracy jest przedstawienie metody grupowania dokumentów XML, która umożliwi uporządkowanie dokumentów XML w oparciu o ich strukturę. Inspiracją zastosowanego podejścia była praca (patrz Zaki 2003) i algorytm TreeMiner, wyszukujący poddrzewa osadzone w kolekcji drzew uporządkowanych. Zaproponowano wykorzystanie miary podobieństwa dwóch drzew jako stosunek wielkości największego poddrzewa wspólnego dla obu drzew, do wielkości większego z obu drzew. Takie podejście wymaga zaproponowania algorytm heurystycznego, do znajdowania maksymalnego poddrzewa wspólnego, ponieważ metody dokładne byłyby tutaj nieefektywne.

2. Grupowanie dokumentów XML

W ostatnim czasie powstało szereg prac na temat eksploracji danych semistrukturalnych, ściślej grupowania dokumentów XML. Istnieje wiele różnych podejść do tego problemu. Kluczowym krokiem jest wybór sposobu reprezentacji dokumentów XML, od którego zależą dalsze postępowania. Dokumenty XML transformuje się do postaci ścieżek (patrz Leung i in. 2005) lub termów (patrz Yoon i in. 2001), albo n-wymiarowego wektora tworzonego na podstawie informacji tekstowych (tagi i/lub zawartość) (patrz Doucet i in. 2002). Powstaje szereg nowych pomysłów, wykorzystujących różne techniki np. reprezentacja dokumentu XML w postaci szeregu czasowego (patrz Flesca i in. 2002). Jednak najbardziej popularnym, najbardziej naturalnym sposobem reprezentacji przetwarzanych dokumentów XML jest postać ukorzenionego, etykietowanego drzewa. (patrz Dalamagas i in. 2004) W takim podejściu istnieje jeszcze rozgraniczenie algorytmów na założenie iż drzewo wejściowe jest nieuporządkowane np. CMTreeMiner, PathJoin (patrz Xiao i in. 2003) czy FreeTreeMiner (patrz Asai i in. 2002) lub drzewo posiada określony porządek, np algorytm FREQT (patrz. Asai i in. 2002) oraz TreeMiner (patrz Zaki 2003). Metoda zaproponowana w poniższej pracy wykorzystuje reprezentację drzewa uporządkowanego, co więcej inspiracją do jej stworzenia był algorytm TreeMiner, który przeszukuje przestrzeń rozwiązań generując kolejno wszystkie rozwiązania. Zaletą TreeMinera w stosunku do innych podejść jest przeszukiwanie zbioru rozwiązań w głąb, co znacznie zmniejsza wymagania pamięciowe. Dodatkowo TreeMiner wyszukuje poddrzewa osadzone, dzięki czemu można go wykorzystać do rozwiązywania problemu grupowania danych semistrukturalnych.

3. Sformułowanie problemu

Dane wejściowe (dokumenty XML) reprezentowane są w postaci uporządkowanych drzew etykietowanych. Zakładamy, że istnieje algorytm A, który dla zbioru etykietowanych drzew uporządkowanych $D = \{T_1, T_2, ...T_n\}$ oraz minimalnego wsparcia φ_{min} wyszukuje wszystkie poddrzewa częste $freq_{\varphi_{min}}(D)$ oraz częste poddrzewa maksymalne *Max(D)*.

Definicja 1. *Drzewem T określa się skierowany graf acykliczny, z jednym wyróżnionym wierzchołkiem, bez poprzedników, korzeniem. T=(V,E,r), V - zbiór*

wierzchołków, E - zbiór krawędzi, r ∈ V - korzeń. V(T),E(T),r(T) oznaczają odpowiednio: zbiór wierzchołków, zbiór krawędzi oraz korzeń drzewa T. Ścieżką pomiędzy wierzchołkami x i y nazywamy najkrótszy ciąg wierzchołków $z, v_1, v_2, ...v_n, y \in V$ *taki, że* $(x, v_1), (v_1, v_2), ..., (v_n, y)$*. Długość ścieżki jest określana jako liczba wierzchołków tworzących ścieżkę. Wysokość drzewa h(T) określa się jako długość najdłuższej ścieżki w drzewie. Wielkością drzewa /T/ nazywa się liczbę jego wierzchołków /T/=/V(T)/.*

Definicja 2. *Drzewem etykietowanym T nazywa się drzewo, dla którego określono funkcję etykietującą* $l(n \in V(T)) : V(T) \rightarrow L$*, L - zbiór etykiet. Wierzchołki w tym samym drzewie, jak również wierzchołki różnych drzew mogą posiadać tą samą etykietę.*

Definicja 3. *Drzewem uporządkowanym określa się drzewo T, w którym kolejność dzieci wierzchołka jest ściśle określona.*

Definicja 4. *S jest poddrzewem T, co oznacza się S<T, jeżeli istnieje funkcja* $\Phi : V(S) \rightarrow V(T)$*, taka że* $(x, y) \in E(S) \rightarrow (\Phi(x), \Phi(y) \in E(T))$*. T jest naddrzewem (superdrzewem) S. Zbiór wszystkich poddrzew drzewa T oznacza się jako Sub(T), zbiór wszystkich nadrzew jako Sup(T).*

Definicja 5. *Poddrzewo S jest poddrzewem częstym w zbiorze D, jeśli* $\sigma_D(S) > \sigma_{min}$*, gdzie* σ_{min}*jest parametrem. Zbiór poddrzew częstych oznaczany jest freq(D), a dla określonego* $\sigma_{min} : freq_{\sigma_{min}}(D)$*.*

Definicja 6. *Poddrzewo S jest poddrzewem maksymalnym w zbiorze D, jeśli*

$$\neg\exists_{T \in Suo_e(S)} \sigma_D(T) > \sigma_{min} \tag{1}$$

Zbiór poddrzew maksymalnych w zbiorze D oznacza się jako Max(D).

Twierdzenie 1. *Dany jest zbiór etykietowanych drzew uporządkowanych.*$D = \{T_1, T_2, ...T_n\}$ *Każde drzewo reprezentuje jeden dokument XML. Zbiór ten należy podzielić na rozłączne podzbiory* $d_1 \cup d_2 \cup ...d_m = D$ *w taki sposób, aby w każdym podzbiorze znalazły się drzewa podobne do siebie, natomiast podobieństwo drzew z różnych podzbiorów było jak najmniejsze.*

Metoda opiera się na definicji miary podobieństwa dokumentów (miary odległości):

$$d(T_1, T_2) = \frac{max\{|M| : M \in freq_2(T_1, T_2)\}}{max(|T_1|, |T_2|)} \tag{2}$$

Miara ta definiuje podobieństwo dwóch drzew jako stosunek wielkości największego poddrzewa wspólnego dla obu drzew, do wielkości większego z obu drzew. Tak użyta miara umożliwia zastosowanie do grupowania algorytmów znanych dla klasycznych problemów eksploracji. W naszym eksperymencie skorzystamy z algorytmu grupowania AHC, który umożliwia dużą eastyczność, a zarazem kontrolę nad ilością i wielkością otrzymywanych klastrów.

4. K-Tree(n) - znajdowanie największego maksymalnego poddrzewa

Przeprowadzone badania algorytmów wyszukujących poddrzewa częste, wykazały ich wykładniczą złożoność w funkcji wielkości badanych drzew. Złożoność ta dyskwalifikuje przedstawione algorytmy, w ich praktycznych zastosowaniach w przetwarzania drzew, składających się z więcej niż kilkudziesięciu wierzchołków. W tej sekcji zostanie przedstawiona heurystyka wyszukująca największą strukturę maksymalną dla zbioru drzew. Ponieważ wykładnicza złożoność przebadanego algorytmu nie wynika z jego budowy, ale z liczby poddrzew częstych, które stanowią rozwiązanie i które algorytm musi odnaleźć, nie ma możliwości opracowania efektywnych dokładnych algorytmów wielomianowych wyszukujących wszystkie struktury częste. Wykładnicza złożoność oznacza również, że nawet wielokrotne zwiększenie wydajności technologii stosowanej do rozwiązania problemu, nie sprawi, że proponowane algorytmy będzie można wykorzystać w praktyce. Dlatego też, aby móc eksplorować większe struktury, w wydajny sposób, należy inaczej zdefiniować problem poszukiwania struktur częstych.

Rozważmy problem znajdowania największego poddrzewa maksymalnego:

$$S_{max}(D) = S \in Max(D) : s(S) = max_{U \in Max(D)}(s(U)) \tag{3}$$

Problem ten znany jest jako NP-trudny (patrz Kilpelainen, Mannilla 1995), dlatego zaproponowano heurystykę wyznaczania największego maksymalnego poddrzewa (maKsymalne drzewo spośród n drzew - K-Tree(n)). Heurystyka jest oparta o wspomniany wcześniej algorytm TreeMiner. O jego przydatności zadecydowała wygodna reprezentacja drzew w postaci list zasięgu, co umożliwia wydajne przeglądanie struktury drzew.

Zaproponowana metoda K-Tree(n) buduje drzewo według zasad podobnych do TreeMinera. Największy nacisk położony jest na minimalizowanie liczby tworzonych poddrzew. W idealnym przypadku, algorytm powinien zbudować największe poddrzewo maksymalne, w każdym kroku dodając jeden wierzchołek do budowanego rozwiązania. Ograniczenia liczby przeszukiwanych struktur w algorytmie TreeMiner, można dokonać w dwóch miejscach. Pierwszym z nich jest moment tworzenia nowej klasy prefiksu, drugim faza łączenia list zasięgu elementów klasy prefiksowej.

4.1. Wybór klasy prefiksowej

W algorytmie *TreeMiner*, dla każdej klasy prefiksu P_n, składającej się z $m = |P_n|$ drzew, tworzonych jest m nowych potencjalnych klas prefiksowych P_{n+1}. Z każdą z m nowo powstałych klas wykonywana jest próba połączenia wszystkich m elementów klasy P_n. W celu ograniczenia liczby tworzonych struktur, w proponowanej heurystyce, tylko na podstawie jednego z m elementów klasy P_n, tworzona jest klasa P_{n+1}. Wybór elementu, który zostanie dołączony do budowanego rozwiązania, tworząc równocześnie klasę P_{n+1}, dokonywany jest na podstawie funkcji oceny $f(R \in P_n)$. Zadaniem funkcji oceny jest wybór kandydata, który umożliwi zbudowanie największego poddrzewa. Aby oceny kandydatów dokonać w sposób efek-

Tabela 1. Przykładowa lista zasięgu

tid	0	1	0	3	1	3	0	1	1
[l,u]	[2,3]	[7,7]	[3,3]	[3,6]	[5,6]	[7,8]	[5,7]	[4,9]	[8,8]

tywny, heurystyka przyjmuje uproszczenie, traktując każdą pozycję listy zasięgu jako trójkę *(tid,[l,u])*, gdzie *tid* - identyfikator drzewa, *[l,u]* - zasięg (jednocześnie l jest indeksem wierzchołka w drzewie T). W takim ujęciu pozycja listy reprezentuje konkretny wierzchołek, w określonym drzewie. Uproszczenie to oznacza pominięcie w rozważaniach sygnatury dopasowania. Strategia heurystyki opiera się na założeniu, że najlepszym kandydatem jest element klasy P_n, przez wybranie którego, powstanie klasa P_{n+1}, której listy zasięgu elementów będą zawierały najwięcej wierzchołków dla każdego drzewa. Inaczej mówiąc, dla każdego drzewa $T \in D$ należy określić ile wierzchołków tego drzewa znajduje się na listach zasięgu elementów nowej klasy P_{n+1}. Podejście takie zakłada, że jeśli wierzchołek znajduje się na jakiejś liście, to być może zostanie w późniejszym etapie dołączony do rozwiązania. Jeśli wierzchołek nie znajdzie się na żadnej liście, to z pewnością nie znajdzie się w rozwiązaniu. Patrząc na problem jeszcze inaczej, proponowana heurystyka dąży do wybrania elementu, który zminimalizuje liczbę wierzchołków, jakie zostaną odrzucone w fazie łączenia list zasięgu. Ponieważ koszt wygenerowania każdej, możliwej na danym etapie klasy i policzenia wierzchołków, które znajdują się na listach zasięgu, jest zbyt duży, heurystyka próbuje oszacować, który z elementów pozwoli zbudować najlepszą klasę, przed jej utworzeniem. Oszacowanie to jest możliwe dzięki przyjętemu założeniu, pozwalającemu traktować pozycję na liście zasięgu wyłącznie jako wierzchołek z określonego drzewa *(tid)*, o określonym zasięgu *([l,u])*. Przy takim uproszczeniu możemy przyjąć, że dowolna pozycja z listy zasięgu e_n nie będzie odrzucona w momencie łączenia list zasięgu, jeśli na liście zasięgu wybranego elementu znajdzie się pozycja *e:e.l<en.l*. Kontynuując rozumowanie można przyjąć, że element mający na swojej liście zasięgu wierzchołki o minimalnych indeksach, gwarantuje, że niewiele wierzchołków zostanie odrzuconych w fazie łączenia list. Strategia taka spowoduje zbudowanie maksymalnego drzewa dla |D|=1. W tym przypadku do budowanego rozwiązania będą dołączane wierzchołki o kolejnych indeksach, zgodnie z porządkiem DFS, aż w rozwiązaniu znajdą się wszystkie wierzchołki wejściowego drzewa. Jednak dla |D|>1 okazuje się, że dla każdego drzewa optymalny element może być inny. Mamy tu klasyczny przypadek wielokryterialnego problemu decyzyjnego. Proponowana funkcja oceny szuka kompromisu wychodząc z założenia, że poddrzewo musi być zawarte w każdym drzewie zbioru D. Dlatego też jakość (czyli wielkość odnalezionego poddrzewa) jest ograniczona od góry przez najgorszy przypadek w zbiorze D. Tym najgorszym przypadkiem jest drzewo, gdzie najmniejszy indeks wierzchołka jest największy pośród wszystkich drzew. Dlatego jako ocena elementu, brany jest największy indeks wierzchołka spośród indeksów minimalnych, dla wszystkich drzew na liście zasięgu danego elementu.

Tabela 1 przedstawia przykładową listę zasięgu pewnego elementu (na liście nie

została przedstawiona sygnatura dopasowania). Zaznaczono wierzchołki o minimalnych indeksach dla każdego drzewa. Oceną elementu jest największy spośród zaznaczonych indeksów, czyli 4 (przypadek dla drzewa 1).

4.2. Wycofanie i dywersyfikacja

Przedstawiona strategia wyboru elementu do rozszerzenia klasy P_n wymaga wprowadzenia oceny przyjętego rozwiązania. Następnie, jeśli ocena ta okaże się niezadowalająca, algorytm powinien wycofać się z podjętej decyzji. Na podstawie funkcji oceny elementu klasy prefiksowej zdefiniowano funkcję oceny klasy prefiksowej $f(R \in P)$:

$$F(P_n) = min_{R \in P_n}(f(R)),\ dla\ P_n \neq \phi \tag{4}$$

$$F(P_n) = max_{T \in D}(s(T)),\ dla\ P_n = \phi \tag{5}$$

(dzięki takiej definicji, klasa której nie można już rozszerzać, zawsze będzie miała najgorszą ocenę) Funkcja oceny F wykorzystana zostanie do oceny trafności wyboru elementu, po utworzeniu nowej klasy.

Element dywersyfikacji heurystyki realizowany jest przez zdolność algorytmu do wycofania. Dywersyfikacja ma na celu poszerzenie przestrzeni analizowanych rozwiązań, aby uniknąć uwięzienia w minimum lokalnym. W proponowanej heurystyce dywersyfikacja realizowana jest poprzez możliwość wycofania algorytmu z podjętej decyzji. Po wyborze elementu i zbudowaniu klasy P_{n+1}porównywane są oceny $F(P_n)$i $F(P_{n+1})$. Wycofanie polega na ponownej generacji klasy P_{n+1} począwszy od klasy P_{n+1-d}, gdzie d - głębokość wycofania. Algorytm przechowuje d_{max} ostatnich rozwiązań, aby móc się wycofać. W fazie ponownej generacji klasy P_{n+1} algorytm nie jest wycofywany. Po wygenerowaniu klasy P_{n+1} zapamiętywane jest najlepsze rozwiązanie, wraz z ciągiem klas prefiksowych, które doprowadziły do takiego rozwiązania. Powrót i generacja klasy P_{n+1} wykonywana jest do czasu, gdy przez określoną liczbę prób nie nastąpi poprawa znalezionego rozwiązania. Aby wycofanie algorytmu mogło poprawić rozwiązanie, należy wprowadzić pewną losowość w algorytmie. Losowość ta ma miejsce w momencie wyboru elementu, który zostanie dołączony do budowanego rozwiązania. Funkcja f buduje ranking tych elementów. Dla każdego elementu określana jest waga:

$$w(R \in P_n) = (f_{max} - F(R))^k \tag{6}$$

gdzie f_{max} - to maksymalna wartość f pośród wszystkich elementów, a k to parametr strojący określający poziom dywersyfikacji. Następnie losowany jest element z prawdopodobieństwem:

$$P_e(R \in P_n) = w(R) / \sum_{R \in P_n} w(R) \tag{7}$$

W przedstawionym wcześniej przykładzie zdefiniowane miary wyglądają następująco (dla t=2):

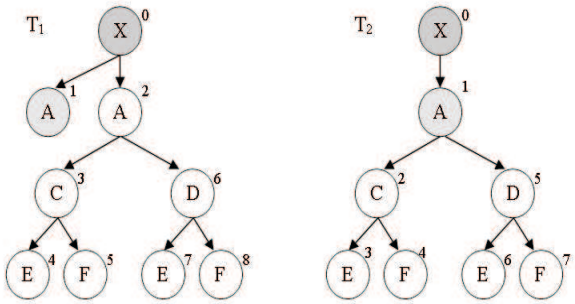
Wybrane w pierwszym kroku rozwiązanie zostanie odrzucone z prawdopodobieństwem 0,83 i algorytm spróbuje poprawić uzyskany wynik. W podanym przykładzie wyborem optymalnym jest element C.

Tabela 2. Przykład

Klasa	$F(P_n)$	$\Delta F(P_n)$	$dF(P_n)$	$P_b(P_n)$
P_1	1			
$P_{2(B)}$	$8(s(T_1))$	7	0,88	0,83
$P_{2(C)}$	4	3	0,38	0,53
$P_{2(E)}$	5	4	0,50	0,63
$P_{2(F)}$	6	5	0,63	0,71
$P_{2(D)}$	7	6	0,75	0,78

Tabela 3. Lista zasięgu elementu A

tid	1	1	2
[l,u]	[1,1]	[2,6]	[1,5]



Rysunek 1. Przykład niewłaściwego rozwiązania

4.3. Wybór wystąpienia

Jak wspomniano wcześniej, w algorytmie TreeMiner istnieją 2 miejsca, w których można ograniczyć liczbę przeglądanych struktur. Wybór jednego elementu z klasy prefiksowej to pierwszy z nich. Jednak z każdym elementem w klasie prefiksowej, związanych może być wiele wystąpień budowanego drzewa, w każdym z drzew zbioru D. Liczba tych wystąpień zależy wykładniczo od rozmiaru drzewa źródłowego, a więc złożoność heurystyki też staje się wykładnicza. Dlatego też, przed fazą łączenia list zasięgu, należy na liście zasięgu wybranego elementu wybrać najlepiej rokujące wystąpienia budowanego rozwiązania w każdym drzewie. Przedstawiana heurystyka zostawia tylko jedno wystąpienie budowanego rozwiązania w każdym drzewie. Jest ono wybierane zgodnie ze strategią obraną przy wyborze elementu klasy prefiksowej. Oznacza to, że dla każdego drzewa ze zbioru D na liście zasięgu jest zostawiane wystąpienie, dla którego ostatni wierzchołek ma najmniejszy indeks. Powracając do tabeli 1, która przedstawia przykładową listę zasięgu z oznaczonymi pozycjami reprezentującymi wystąpienie o najmniejszym indeksie, w tym kroku usuwamy z listy pozostałe elementy. Podobnie jednak, jak przy wyborze elementu, podejście to nie jest optymalne, dlatego algorytm dopuszcza wybór innej pozycji z pewnym prawdopodobieństwem. Poniższy przykład przedstawia sytuację, gdy takie podejście powoduje, że nie udaje się odnaleźć właściwego rozwiązania.

Zgodnie z przedstawionymi wcześniej zasadami wybrany zostanie element A. Jednak element A posiada 2 wystąpienia w drzewie T1.

W fazie wyboru wystąpienia na przedstawionej liście zasięgu pozostaną tylko

oznaczone pozycje, w wyniku czego powstała klasa $P_{2(A)}$nie będzie posiadała żadnych elementów (przy założeniu $\sigma_{min} = 2$). Aby uniknąć takiego problemu, przy wyborze wystąpienia zastosowano podobne podejście, jak przy wyborze elementu. Z pewnym prawdopodobieństwem algorytm potrafi wybrać rozwiązanie potencjalnie gorsze. Wagi rozwiązań użyte do losowania są tworzone podobnie, jak przy wyborze elementu klasy prefiksowej. Dla każdego drzewa losowana jest jedna pozycja z listy zasięgu. Wagi pozycji dla elementu R i drzewa T tworzone są następująco:

$$w(e \in sl(R)) = (l_{maxT=e.T} - e.l)^k \tag{8}$$

gdzie *sl(R)* to lista zasięgu elementu R, l_{max_r} - największy indeks wierzchołka w drzewie T na liście zasięgu *sl(R)*. Pozycja losowana jest z prawdopodobieństwem:

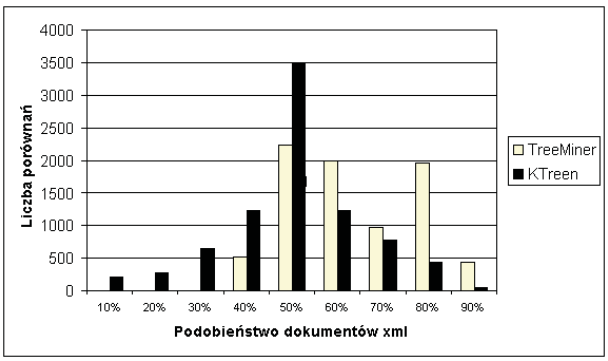
$$P_p(e \in sl(R)) = w(e) / \sum_{g \in sl(R):g.T=e.T} w(g)) \tag{9}$$

5. Eksperyment

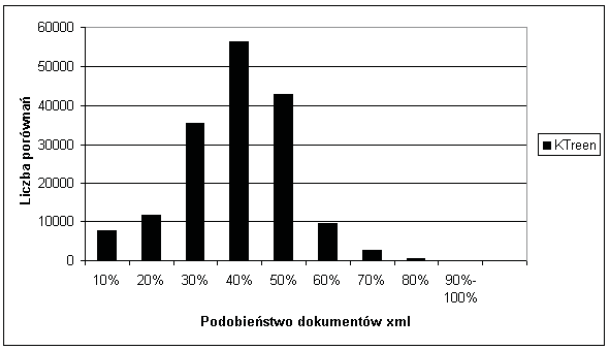
Eksperyment przeprowadzono na danych ze zbioru INEX (patrz xmlmining.lip6.fr). Zbiór dokumentów XML zawiera dane rzeczywiste pochodzące ze zbioru WIKIPEDIA. Eksperyment został przeprowadzony na trzech instancjach dokumentów. Zbiór 128 dokumentów o ilości węzłów poniżej 16, druga instancja liczy 578 dokumentów o węzłach do 20 wierzchołków, oraz trzecia instancja składa się ze zbioru 932 dokumentów powyżej 22 wierzchołków. Podobieństwo było sprawdzane na poziomie dokumentu. Porównywano każdą parę dokumentów w zbiorze. Do testów wykorzystano komputer Toschiba, system Win XP sp.2 z procesorem Intel Centrino Duo 2.16GHz oraz 1 GB RAM. Badania przeprowadzono w dwóch etapach. Celem pierwszego etapu było sprawdzenie efektywności miary wyznaczającej podobieństwo dokumentów xml. Dokonano porównania otrzymanych wyników uzyskanych przez heurystykę z algorytmem dokładnym TreeMiner. Dla każdej instancji heurystyka została uruchomiona pięciokrotnie, a przedstawione wyniki zostały uśrednione. Drugi etap polegał na wykorzystaniu otrzymanego podobieństwa w zadaniu grupowania dokumentów xml.

5.1. Podobieństwo dokumentów

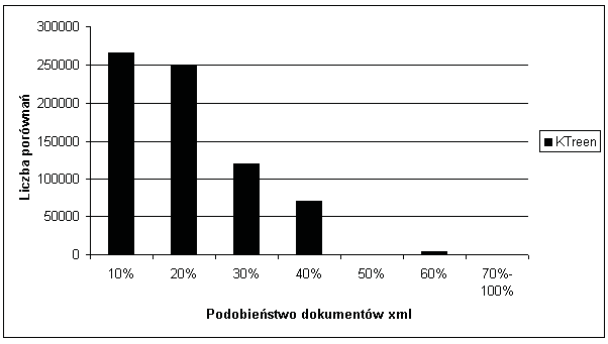
Jak wcześniej zostało wspomniane celem proponowanej heurystyki było wyznaczenie podobieństwa dokumentów pod względem struktury. W pierwszym etapie eksperymentu badaliśmy jak efektywna jest nasza miara określająca podobieństwo. Badania zostały przeprowadzone na trzech instancjach (128, 578 oraz 932 dokumenty). Poniżej (Rysunek 6-8) przedstawiono zestawienia otrzymanych wyników. Porównujemy tutaj podobieństwo dokumentów otrzymane dla każdej z instancji wyznaczone przez heurystykę oraz dla porównania przez algorytm dokładny. Porównujemy każdą parę dokumentów w zbiorze, czyli poszukujemy największego wspólnego poddrzewa porównując każdy dokument z każdym. W badaniach można



Rysunek 2. Instancja 1 - 128 dokumentów



Rysunek 3. Instancja 2 - 578 dokumentów



Rysunek 4. Instancja 3 - 932 dokumenty

zauważyć,że już dla drugiej instancji zawierającej powyżej 16 wierzchołków algorytm dokładny okazał się zawodny, dlatego na wykresach 7 i 8 pojawia się tylko wynik heurystyki.

Otrzymane wyniki potwierdzają wcześniejsze stwierdzenie, że dla danych rzeczywistych, które często zawierają kilkadziesiąt wierzchołków algorytm dokładny jest niewystarczający. Heurystyka, mimo mniejszej dokładności spełnia swoje zadanie. Można łatwo zauważyć, że dla dokumentów o większych rozmiarach podobieństwo znacznie spada. Być może należałoby sprawdzić jeszcze na innym zbiorze danych, o bardziej zróżnicowanej strukturze niż zbiór Wikipedii. Jednocześnie też zastanowić się nad słabymi punktami heurystyki.

5.2. Grupowanie dokumentów

Do zadania grupowania kolekcji dokumentów wykorzystano metodę opartą na metodzie grupowania AHC. Algorytm został zaimplementowany w PLSQL. Danymi wejściowymi są podobieństwa pomiędzy parami dokumentów wraz z identyfikatorami dokumentów. Początkowo każdy z N dokumentów tworzy osobny klaster. Następnie w kolejnych krokach (poziomach) łączymy ze sobą 2 najbliższe klastry, aż do momentu kiedy wszystkie dokumenty znajdą się w jednym klastrze. Wybraną metodą łączenia klastrów jest metoda pełnego łączenia(complete linkage). Jako podobieństwo pary klastrów przyjmujemy najniższe podobieństwo z wszystkich podobieństw pomiędzy dokumentami gdzie jeden z dokumentów należy do jednego klastra, a drugi dokument do następnego klastra. Po obliczeniu podobieństw pomiędzy wszystkimi parami klastrów łączymy ze sobą tę parę klastrów, która jest do siebie najbardziej podobna (ma najwyższe podobieństwo). Jeśli dwie pary klastrów są do siebie równo podobne, to łączymy tą parę, która po połączeniu utworzy klaster o mniejszej liczbie dokumentów. Po połączeniu przechodzimy do następnego kroku (poziomu), w którym próbujemy znowu połączyć ze sobą 2 klastry. Procedurę powtarzamy do momentu gdy połączenie nie będzie możliwe, czyli do uzyskania 1 klastra. Dla każdego poziomu, jako podobieństwo poziomu określamy podobieństwo pomiędzy łączonymi klastrami. Możemy określić dopuszczalny minimalny próg podobieństwa z jakim mogą być łączone klastry. Wtedy jako wynik klastrowania przyjmujemy klastry otrzymane na poziomie, gdzie podobieństwo poziomu nie spadło poniżej obranego progu. Można też obserwować pomiędzy którymi poziomami następuje duży skok pomiędzy podobieństwami i jako wynik przyjąć klastry z poziomu „przed" wystąpieniem skoku. Trzecim wskaźnikiem wyboru klastrów wynikowym jest liczba otrzymanych klastrów. Wybieramy poziom, w którym liczba utworzonych klastrów spełnia nasze wymagania.

W tabelach 4 i 5 przedstawiono wyniki grupowanie dla instancji 128 i 932 dokumenty.

Algorytm dokładny wykazał, że dla wszystkich 932 drzew istnieje wspólne poddrzewo 5 elementowe, a podobieństwo całego zbioru 932 wynosi 0.217. Jest to minimalne podobieństwo jakie będą zawierały dowolnie utworzone klastry na tym zbiorze. Jeśli przyjąć że, poddrzewo nie musi się znajdować we wszystkich dokumentach zbioru a zaledwie w 80% (wsparcie)procentach dokumentów to rozmiar maksymalnego poddrzewa znalezionego przez algorytm dokładny wynosi 9 a podo-

Tabela 4. Sprawdzenie jakości klastrowania poprzez wyszukanie algorytmem dokładnym podobieństwa zbioru 128 dokumentów

klaster	liczność	minsup*	sup**	realsup***	r.poddrzewa	podobieństwo
1	28	16	18	64%	6	0.429
2	18	10	13	72%	10	0.714
3	14	8	8	57%	10	0.714
4	7	4	7	100%	7	0.5
5	10	6	6	60%	9	0.643
6	5	3	3	60%	8	0.5
7	11	6	7	64%	10	0.769
8	23	13	16	70%	6	0.6
9	5	3	4	80%	10	0.625
10	7	4	4	57%	11	0.917

* minsup - liczba dokumentów, w których powinno pojawić się poddrzewo
** sup - liczba dokumentów, w których rzeczywiście poddrzewo się pojawiło
*** realsup - rzeczywiste wsparcie

bieństwo zbioru wzrasta do 0.391. Przy wsparciu 69% poddrzewo i podobieństwo wynoszą odpowiednio 10 i 0.435. Przy niższym wparciu algorytm dokładny wyczerpuje całą dostępną pamięć i nie wylicza się.

Zbiór Inex był podzielony na 21 kategorii tematycznych. W naszych badaniach sprawdziliśmy czy istnieje związek pomiędzy strukturalnym a tematycznym podobieństwem. Najpierw zbiór 932 został podzielony na klastry według przynależności do 21 kategorii tematycznych. Następnie dla każdego klastra zostało znalezione maksymalne poddrzewo. W każdym klastrze istnieje poddrzewo o rozmiarze 5. Uznajemy, że jeśli nie zostanie znalezione zauważalnie większe poddrzewo dla danego klastra to nie ma związku pomiędzy strukturą a zawartością. Heurystyka wykazała brak związku. Zostały wykonane 3 uruchomienia. Znalezione poddrzewa wahały się od 1 do 8. Algorytm dokładny również wykazał brak związku. Na 21 klastrów, aż 12 wykazało największe poddrzewo o rozmiarze 5. 1 klaster miał poddrzewo o rozmiarze 7 i podobieństwo wewnętrzne równe 0.412. 4 klastry miały poddrzewo 9 a ich podobieństwo wahało się granicach 0.38 – 0.45. 2 klastry : 10 i 0.55. i 2 klastry: 13 i 0.56-0.812.

6. Wnioski i uwagi końcowe

W pracy przedstawiono technikę grupowania dokumentów xml na podstawie ich strukturalnego podobieństwa. Zaproponowano miarę definiującą podobieństwo dwóch drzew jako stosunek wielkości największego poddrzewa wspólnego dla obu drzew, do wielkości większego z nich. Do wyznaczenia maksymalnego poddrzewa opracowano i wykorzystano metodę heurystyczną. Otrzymane w ten sposób podobieństwo wykorzystano w grupowaniu kolekcji dokumentów xml. Przeprowadzone badania pokazały, że stosowana miara spełnia swoje zadanie. W sytuacji gdy zawodzi metoda dokładna heurystyka pozwala na przeprowadzenie badań. Jednak można zaobserwować również jej słabe strony. Dla dokumentów o większych roz-

Tabela 5. Sprawdzenie jakości klastrowania poprzez wyszukanie algorytmem dokładnym podobieństwa zbioru 932 dokumenty

klaster	liczność	minsup*	sup**	realsup***	r.poddrzewa	podobieństwo
1	77	46	53	69%	13	0.591
2	71	42	47	66%	15	0.609
3	48	28	29	60%	13	0.591
4	32	19	20	63%	14	0.609
5	24	14	15	63%	15	0.652
6	516	309	417	81%	9	0.429
7	34	20	20	59%	16	0.652
8	49	29	32	65%	15	0.609
9	23	13	15	65%	13	0.656
10	58	34	39	67%	14	0.591

* minsup - liczba dokumentów, w których powinno pojawić się poddrzewo
** sup - liczba dokumentów, w których rzeczywiście poddrzewo się pojawiło
*** realsup - rzeczywiste wsparcie

miarach (większej liczbie węzłów), czas wykonywania wzrasta wykładniczo, natomiast podobieństwo znacznie maleje. Ma to również odzwierciedlenie w procesie grupowania, im mniej zróżnicowane wartości miary podobieństwa (np. dla instancji 932 dokumenty) tym gorsza jakość otrzymanych klastrów. Jak zostało również wspomniane wcześniej, proponowane rozwiązanie nie nadaje się do wykorzystania w grupowaniu tematycznym dokumentów. Otrzymane wyniki wykazały, iż w takim rozumowaniu nie ma związku pomiędzy strukturą a zawartością dokumentu. Dalsze badania będą prowadzone w kierunku usprawnienia metody (np. redukcji liczby porównań dokumentów). Heurystykę można wykorzystać w różnych zastosowaniach. Jednym z nich może być jej modyfikacja, która polegałaby na wyszukiwaniu poddrzew, które byłyby częste w określonym zbiorze drzew, a nie występowały w pozostałych drzewach. Poddrzewa takie można by wykorzystać do budowy wydajnego klasyfikatora.

Literatura

Asai, T., Abe, K., Kawasoe, S., Arimura, H., Satamoto, H. and Arikwa, S., (2003) Efficient substructure discovery from large semi-structured data. *IIn Proc. of the 2nd SIAM Int. Conf. on Data Mining (SDM'02).*

Dalamagas, T., Cheng, T., Winkiel, K. and Sellis, T. (2004) Clustering XML documents by structure. *EDBT Workshop*, pp.547-556.

Doucet, A. and Ahonen-Myka, H. (2002) Naive clustering of a large XML document collection. *In Proceedings of the First Annual Workshop of the Initiative for the Evaluation of XML retrieval (INEX), ERCIM Workshop Proceedings.* Schloss Dagstuhl, Germany, 81-88.

Flesca, S., Manco, G., Masciari, E., Pontieri, L. and, Pugliese, A. (2002) Detecting structural similarities between XML documents. *In Proc. 5th Int. Workshop on the Web and Databases (WebDB '02).* Madison, Wisconsin.

Kilpelainen, P. and Mannia, H. (1995) Ordered and unordered tree inclusion. *SIAM Journal on Computing* **24**.

Kaufman, L. and Rousseeuw, P.J. (1990) Finding Groups in Data: An Introduction to Cluster Analysis. *ISBN-13: 978-0-471-73578-6, John Wiley & Sons.*

Liu, J., Wang, J.T.L., HSU, W. and Herbert, K.G (2004) XML clustering by Principal Component Analysis. *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004).*

Termier, Z., Rousset, M.C. and Sebag. M (2002) Treefinder: a first step towards xml data mining. *IEEE International Conference on Data Mining (ICDM02)*, 450-457.

Xiao, Y., Yao, J-F., Li, Z. and Dunham, M., (2003) Efficient data mining for maximal frequent subtrees. *In Proc. of the 2003 IEEE Int. Conf. on Data Mining (ICDM'03).*

Yang, R., Kalnis, P.and Tung, A.K.H.(2005) Similarity Evaluation on Tree-structured Data. *ACM SIGMOD Baltimore, Maryland, USA.*

Yoon, J.P., Raghavan, V., and Chakilam, V., (2001) BitCube: A three-dimensional bitmap indexing for XML documents. *SSDBM , Fairfax, Virginia*, 158-167.

Zaki, M.J., (2002) Efficiently mining frequent trees in a forest. *In 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

Zaki, M.J. and Aggarwal, C.C., (2003) XRules: An effective structural classiffier for XML data. *In Proc. of the 2003 Int. Conf. Knowledge Discovery and Data Mining (SIGKDD'03).*

K-Tree(n) - clustering XML documents

The article presents the method of clustering XML documents, based on the representation of XML documents as an ordered, labeled tree. The method proposed orders the collection of XML documents, including only the structure of the documents without the content. The authors propose similarity measure as a relation of the maximal common subtree to the bigger one. The authors propose heuristic algorithm K-Tree(n) to find maximal common subtree. For clustering the modification of method AHC was chosen. In this method the distance between documents was replaced by similarity, appointed by heuristics.