Michał Barański, Adam Pelikant
Wyższa Szkoła Informatyki i Umiejętności
email: adam_pelikant@wsinf.edu.pl

WYSZUKIWANIE PODCIĄGU W CIĄGU Z ZASTOSOWANIEM OBIEKTÓW CLR

Streszczenie – Celem niniejszej pracy jest zaprezentowanie możliwości zastosowania mapowanych na elementy proceduralne Transact SQL klas obiektowych CLR tworzonych na platformie .NET w złożonych algorytmach przetwarzania. Przedstawione zostały podstawy teoretyczne algorytmów dopasowania łańcuchów dla alfabetów skończonych. Dla wprowadzonych alfabetów nieskończonych rozwiązania te nie mogą być w sposób prosty zmodyfikowane, dlatego zaproponowany został algorytm DTW (Dynamic Time Warping), który został oprogramowany z zastosowaniem reguł mapowania do obiektów rozszerzenia proceduralnego SQL. Przedstawiono elementy praktycznej realizacji praktycznej oraz dokonano omówienia wyników eksperymentu numerycznego dopasowującego gesty.

Słowa kluczowe: dopasowanie do wzorca, dynamic time warping, DTW, programowanie CLR

1 Wstęp

W wielu problemach praktycznych mamy do czynienia z problemem wyszukiwania fragmentu informacji w szerokiej puli danych. Takie zadania możemy określić mianem problemu dopasowania do wzorca. Pojawia się on bardzo często w zagadnieniach związanych z przetwarzaniem tekstu. W najbardziej elementarnej formie będzie miał postać znalezienia podciągu znaków w ciągu znaków, albo wykrycia które ciągi spośród dużego ich zestawu zawierają w sobie ten podciąg. Dla tak postawionego problemu należy zdefiniować pewne pojęcia podstawowe [1].

Definicja 1. Alfabetem nazywamy zbiór znaków możliwych do zakodowania w ustalonym standardzie np. UNICODE. Alfabet będziemy oznaczać symbolem Σ.

Dla takiej definicji zestaw znaków poniżej jest elementem alfabetu, zastosowano apostrof jako ogranicznik napisu:

’a’, ’b’, ’c’, ’#’, ’@’, ’β’, ’λ’, ’0’, ’1’, ’2’ $\in \Sigma$;

natomiast poniższy zestaw, stanowi zbiór elementów nie należących do alfabetu

’abc’,’#@&’, 'Jan’ $\notin \Sigma$

W zbiorze Σ możemy wyróżnić podzbiór znaków białych znaków, tzn. takich które nie mają reprezentacji graficznej np. spacja, tabulacja etc. Zbiór ten oznaczmy przez B i wykorzystujemy do definiowania przerw między słowami.

Definicja 2. Tekstem nazywamy niepusty ciąg znaków alfabetu i oznaczamy przez T.

Przykłady tekstów to:

’abc’,’Ala ma Asa. As to pies.’, ’123’, ’345 234’

Definicja 3. Słowem nazywamy niepusty ciąg znaków alfabetu Σ niezawierający białych znaków. Zbiór wszystkich możliwych słów oznaczamy przez Θ:

$\Theta = (\Sigma \setminus B)^* \setminus \{\varnothing\}$

Do zbioru Θ należą wszystkie możliwe ciągi utworzone ze znaków niebędących białymi, z wyjątkiem ciągu pustego.

Przykłady słów:

’abc’,’#@&’, 'Jan’, ’123’ $\in \Theta$

’Ala ma Asa’, ’345 234’, ’ ’, ’’ $\notin \Theta$

Rozszerzeniem pojęcia słowa jest jednostka nazwana, którą możemy zdefiniować jako.

Definicja 4. Jednostką nazwaną nazywamy niepusty ciąg słów, któremu przypisane jest jedno wspólne znaczenie, etykietę, które nazywamy Typem.

Przykłady jednostek nazwanych to:

’Jan Kowalski’, Pracownik

’22.01.2014’, DataPublikacji

’51.869N 16 _19.410E’, LokalizacjaObiektu

W wielu przypadkach mamy do czynienia z mało licznymi słownikami, klasycznym przykładem jest opis łańcucha DNA [7], [8], [9] przez symbole nukleotydów (ściślej zasad azotowych) adeniny, guaniny, cytozyny i tyminy. Słownik składa się z czterech znaków A G C T. Podobnie białka składają się z sekwencji 20 aminokwasów kodowanych od A do Y (z pominięciem J) lub opisywanych przez 60 kodonów czyli trójki nukleotydów A G C U (tyminę zastępuje uracyl) [10], [11], [12]. Mniej znanym przykładem jest notacja muzyczna ABC [4], w której podstawowy, obejmujący dwie oktawy słownik składa się z symboli: C D

E F G A B c d e f g a b. W przypadku kodowania binarnego słownik składa się z dwóch symboli 0 1. Przytoczone powyżej definicje są wykorzystywane w zadaniach automatycznego tłumaczenia [2], [3] lub wyszukiwania informacji w plikach tekstowych [5], [6].

Jednak w wielu zadaniach praktycznych nie występują w zapisie znaki białe separatory. Możemy mówić, że do notacji używamy bardzo długich słów, w takim przypadku posługujemy się raczej pojęciem łańcucha, który możemy definiować tak jak poprzednio słowo, lub korzystając z rekurencji alfabetem Σ:

- ε jest pustym łańcuchem (niezawierającym żadnego znaku) nad alfabetem Σ;
- jeśli x jest łańcuchem nad Σ i $a \in \Sigma$ to wyrażenie ax będące ich konkatenacją jest również łańcuchem nad Σ.

Jeśli obiektu nie da się utworzyć w powyższy sposób, nie jest on łańcuchem. Konkatenacją łańcuchów $x = x_1x_2...x_m$ i $y = y_1y_2...y_n$ nazywamy wyrażenie $xy = x_1x_2...x_my_1y_2...y_n$. Spełnia ona następujące trzy warunki:

- posiada element neutralny ε będący łańcuchem pustym ($x\varepsilon = \varepsilon x = x$);
- jest łączna $xyz = (xy)z = x(yz)$;
- nie jest rzemienna $xy \neq yx$.

Możemy postawić sobie zadanie wyszukania wystąpienia w łańcuchu $y = y_1y_2...y_n$ łańcucha $x = x_1x_2...x_m$, przy czym zakładamy, że n>=m. Jako rezultat uzyskamy indeks j wskazujący miejsce pojawienia się pierwszego znaku łańcucha x w y o ile kolejne znaki y będą takie same jak znaki w x. Przy czym j może przyjąć wartość $j \in \{0, m-n+1\}$, 0 oznacza brak dopasowania. Rozwiązaniem tak postawionego problemu mogą być algorytmy:

- naiwny lub zachłanny (brut force), w którym sprawdzany jest każdy znak łańcucha x z każdym kolejnym elementem łańcucha y rozpoczynając od pierwszego znaku, jeśli na którejś z pozycji porównanie daje wynik nierówny, wskaźnik w łańcuchu y jest przesuwany o jeden aż do momentu uzyskania dopasowania albo chwili gdy wskaźnik w łańcuchu y osiągnie wartość m-n; złożoność obliczeniowa tego algorytmu wynosi O((m-n+1)n);
- Rabina-Karpa, który wykorzystuje wartości funkcji haszującej (skrótu, mieszającej) dla wzorca oraz tekstu, funkcja haszująca jest dowolną zależnością przypisującą elementowi alfabetu liczbę całkowitą, najczęściej stosuje się funkcję o postaci

$h(x)=(x(1)r^{n-1}+x(2)r^{n-2}+...+x(n)) \bmod q$
gdzie x(i) jest kodem kolejnego znaku łańcucha x, r liczbą symboli alfabetu, a q dużą liczbą pierwszą, taką że wyrażenie q(r-1) nie powoduje przepełnienia dla stosowanego języka programowania; porównanie odbywa się na zasadach takich samych jak w przypadku algorytmu naiwnego, ale działa na wartościach funkcji haszującej, algorytm wymaga preprocesingu, którego złożoność obliczeniowa wynosi O(n) natomiast średnia złożoność porównania jest O(m+n) natomiast w najgorszym przypadku może ona osiągnąć wartość pesymistyczną O((m-n+1)n);

- Knutha-Morissa-Pratta, który wykorzystuje tablicę zawierającą informacje o miejscu kolejnej próby dopasowania, która pozwala na przeniesienie kolejnej próby dopasowania nie o 1 ale o skok równy minimum z długości szukanego ciągu i pozycji, na której w przeszukiwanej części znajduje się pierwszy znak z wyszukiwanego łańcucha, złożoność preprocesingu wynosi O(n), a dopasowania O(m);
- Boyera-Moore'a – dopasowanie łańcucha x jest realizowane od ostatniego jego znaku lecz przyłożonego do tekstu y począwszy od jego początku, w chwili znalezienia niezgodności realizowane jest jeden z dwóch wariantów przesunięcia: w dopasowywanym tekście nie znaleziono pierwszego znaku wzorca (bad suffix shift) przesuwamy się o całą długość wzorca, w przypadku przeciwnym (good suffix shift) do wystąpienia dopasowania do pierwszego znaku, w wielu przypadkach umożliwia on długie skok, złożoność preprocesingu wynosi O(m\n), a dopasowania O(m·n);
- Aho-Corasicka – przetwarzanie tekstu y przez automat skończony reprezentujący wzorzec x, podstawą jego działania jest drzewo, w którym poza pustym węzłem nadrzędnym (root) reprezentowane są wszystkie potencjalne słowa wynikające z przyjętego algorytmu, przeszukiwanie drzewa jest realizowane poziomo, złożoność preprocesingu wynosi O(n), a dopasowania O(m);
- Baeza-Yates–Gonnet – polega na zbudowaniu macierzy, w której nagłówkami kolumn są kolejne elementy łańcucha, do którego dopasowujemy wzorzec, którego elementy stanowią

nagłówki wierszy, macierz wypełniana jest odległościami między elementem wzorca i tekstu w ten sposób, że jeśli x(i)=y(j)=>0 w przeciwnym przypadku 1, wzorzec jest wykryty, jeżeli istnieje taka diagonala, na której wszystkie wartości są równe 0;

- z próbkowaniem – dopasowanie całości wzorca po pozytywnym dopasowaniu jego fragmentu (próbki), jeśli próbka jest dopasowana do tekstu kontynuujemy sprawdzanie pozostałego fragmentu wzorca korzystając np. z metody Knutha-Morissa-Pratta złożoność obliczeniowa może być oszacowana na O(m log m), ale jest silnie zależna od wyboru wzorca;
- inne mniej rozpowszechnione.

Wadą takich algorytmów, być może poza Aho-Corasicka oraz Baeza-Yates–Gonnet'a jest fakt, że są w stanie wskazać tylko jedno pełne dopasowanie, nie wykrywają natomiast dopasowani częściowych z pewnym dopuszczalnym maksymalnym marginesem błędu.

Należy również wskazać, że rozważaliśmy do tej pory tylko alfabety skończone. Łatwo możemy wyobrazić sobie alfabet (meta alfabet) nieskończony, który składa się z liczb naturalnych (szerzej całkowitych). W takim przypadku zarówno wartości '1', '5', '9', jak również '11', '213', '54321' są znakami tego alfabetu. Dodatkową cechą charakterystyczną takiego alfabetu jest fakt, że stanowi on zbiór policzalny. W praktycznych zastosowaniach może służyć do kodowania sekwencji liczności zdarzeń.

Możliwe jest również wprowadzenie alfabetów nieskończonych, niepoliczalnych np. takich gdzie znakami są liczby rzeczywiste. Choć formalnie zbiór taki jest niepoliczalny, to ze względu na skończoną reprezentację liczb rzeczywistych w komputerze (najczęściej cztero- lub ośmiobitową) stają się zbiorem policzalnym. W praktyce ma on zastosowanie przy opisie dźwięku (mowa, muzyka) za pomocą współczynników cepstralnych (MFCC lub HFCC) [12], [13]. Może zostać użyty do parametryzacji gestów, ruchu, gdzie wartości wskazują na położenie analizowanych elementów. W przypadku dopasowywania wzorca stosowanie omówionych algorytmów w większości przypadków jest bezużyteczne. Wynika to z faktu, że dla takiej notacji stosowanie ścisłego porównania jest bezzasadne. Interesuje nas możliwie najbliższe podobieństwo do wzorca, a nie ścisła równość. Z omawianych powyżej tylko algorytm Baeza-Yates–Gonnet'a, przy założeniu wprowadzenia w macierzy zamiast wartości {0, 1} wartości podobieństwa <0, 1> oraz automatów skończonych w zmodyfikowanym algorytmie Aho-Corasicka pozwala na dopasowanie nieprecyzyjne. Możemy zastosować funkcję

przynależności znaną z teorii zbiorów rozmytych. Innym, wydajnym rozwiązaniem tego problemu jest wykorzystanie algorytmu DTW (Dynamic Time Warping) [12], [13]. Takie podejście będzie pokazane w tym przypadku. Ze względu na wydajność i cechy zastosowanego środowiska zastosowano mechanizm mapowanych klas CLR.

2 Zastosowanie CLR do budowania elementów proceduralnych

Common Language Runtime (z ang. Środowisko Uruchomieniowe Wspólnego Języka) **CLR** - jest środowiskiem uruchomieniowym dla platformy .NET. Zarządza ono cyklem życia aplikacji, kompiluje kod źródłowy, zarządza pamięcią operacyjną oraz zapewnia bezproblemową współpracę pomiędzy programami napisanymi w różnych językach programowania [[1]5], [16], [17]. Tekst źródłowy programu napisanego w języku wyższego rzędu jest w czasie kompilacji zamieniany na tzw. kod pośredni, który jest interpretowany i przekształcany w kod wykonywalny przez środowisko uruchomieniowe. W przeciwieństwie do innych języków programowania wysokiego poziomu, języki należące do platformy .NET nie tworzą kodu wykonywalnego dla konkretnej architektury, ale właśnie dla CLR. Technologia CLR umożliwia tworzenie następujących obiektów dla serwera Microsoft SQL Server:

- własne typy danych [16], [17],
- funkcje skalarne i tabelarzyczne,
- procedury składowane,
- procedury wyzwalane (triggers),
- funkcje agregujące użytkownika (aggregates) [15].

Wszystkie wymienione elementy są tworzone jako instancje klasy, bądź struktury i kompilowane jako biblioteki DLL, a następnie na serwerze tworzy się z nich elementy pośredniczące tzw. `assemblies`.

Integracja CLR z SQL Server działa tak, że hostem dla CLR jest SQL Server. Ma to pozytywny efekt w postaci uniknięcia konfliktów w dostępie do zasobów jakimi operuje dana instancja a zasobami, którymi operuje system operacyjny serwera. Dzięki temu to silnik bazy danych zarządza takimi procesami jak przyznawanie pamięci obiektom CLR, oczyszczanie pamięci z nieużywanych obiektów, zarządzanie izolacją aplikacji CLR w sposób taki, aby różne aplikacje nie miały wpływu na działanie innych równocześnie działających aplikacji. Dostęp do takiej funkcjonalności oznacza wiele dodatkowych możliwości dla programistów, którzy mogą tworzyć obiekty bazodanowe w oparciu o platformy .NET

Najważniejszymi elementami wchodzącymi w skład środowiska zarządzanego CLR są:

- Garbage Collector – automatycznie zwalnia nieużywaną pamięć;
- Class Loader – mechanizm odpowiedzialny za zarządzanie metadanymi, wczytywanie interfejsów klas;
- Exception Menager – zarządza obsługą wyjątków;
- Kompilator Just-In-Time – konwertuje kod pośredni IL na kod maszynowy;
- CLR Code Acces Security – mechanizmy bezpieczeństwa, zarządzające możliwością uruchamiania kodu w zależności od kontekstu użytkownika;
- Type Safety – odpowiedzialny za nadzorowanie konwersji typów.

Zdefiniujmy klasę wykonującą wybraną operację na dwóch parametrach i zwracającą wynik. Pierwszym jej elementem jest dyrektywa kompilatora *Microsoft.SqlServer.Server.SqlProcedure* określająca docelowy rodzaj obiektu po stronie MS SQL Server. Zdefiniowana w tej klasie metoda statyczna ma trzy parametry dwa pierwsze określające dane wejściowe oraz trzeci wyjściowy przedstawiający wynik. Ciało klasy stanowi instrukcja wyznaczająca wynik.

```
using System;
public class wynik
[Microsoft.SqlServer.Server.SqlProcedure]
public static void resztap(double? a, double? b, out
double? wyn)
{wyn = a \ b;}
}
```

Pozostaje utworzenie typu wiążącego ASSEMBLY, dla którego podajemy nazwę, tryb autoryzacji, w postaci kwalifikowanej nazwy biblioteki *dll* oraz tryb dostępu.

```
CREATE ASSEMBLY funkcja
AUTHORIZATION [dbo]
FROM 'C:\...\funkcja.dll'
WITH PERMISSION_SET = SAFE
```

W ostatnim kroku tworzymy procedurę TSQL, w której liście parametrów następuje rzutowanie typów występujących po stronie bazy danych na typy klasy C# oraz wskazanie przez nazwę kwalifikowaną nazwy zewnętrznej metody. Nazwa ta składa się z nazwy obiektu *ASSEMBLY*, nazwy klasy C# i nazwy metody.

```
CREATE Procedure ResztaP(@a float, @b int, @c float
OUTPUT)
AS EXTERNAL NAME funkcja.wynik.resztap
```

Tak utworzona procedura może być wykorzystywana na takich samych zasadach jak gdyby była napisana z użyciem składni TSQL. Według podobnych zasad możemy utworzyć inne elementy proceduralne serwera bazy danych.

Takie podejście do przetwarzania ma szereg pozytywnych cech. Przede wszystkim możemy stosowa metody niedostępne z poziomu serwera bazy danych, a dostępne z poziomu klas języków wyższego rzędu. Koronnym tego przykładem są operacje związanie z systemem plików i obsługą systemu operacyjnego. Również realizacja złożonego przetwarzania numerycznego jest łatwiejsza ze względu na dostępne dla języków wyższego rzędu metody, biblioteki i narzędzia. Ze względu na kompilacje kodu i jego optymalizację, a także odwołanie do takiej postaci po stronie serwera uzyskiwana jest duża wydajność. Przede wszystkim, przetwarzanie jest wykonywane jest w miejscu składowania danych co powoduje, że eliminowane jest przesyłanie danych ze składu do końcówki klienckiej, co ma również niebagatelny wpływ na wydajność. Integracja silnych narzędzi programistycznych z serwerem bazy danych jest w opinii autorów najskuteczniejszym sposobem uzyskiwania i wdrażania silnych algorytmów przetwarzania. Ponadto dzięki zastosowaniu mechanizmu UDT (User Data Types) pozwala na składowanie złożonych typów wraz z dedykowanymi dla nich metodami po stronie bazy, co prowadzi do wprowadzenia obiektowości po stronie serwera.

3 Algorytm DTW – nieliniowe dopasowanie czasowe

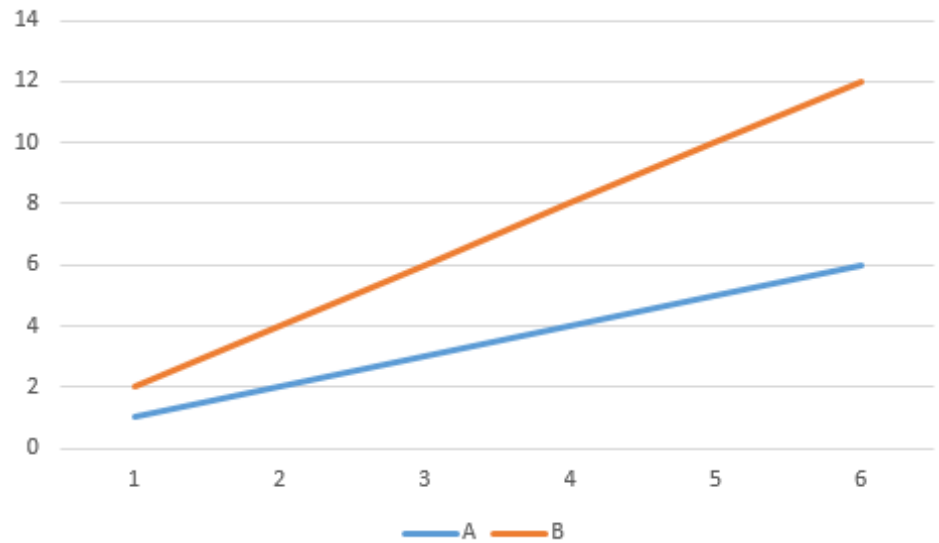
Nieliniowa transformacja czasowa DTW (Dynamic Time Warping) polega na sprawdzaniu podobieństwa badanej sekwencji z sekwencją stanowiącą model, dopuszczając przy tym nieliniowe transformacje czasu dla obu badanych serii danych. Dokładniej mówiąc główną cechą tego rozwiązania jest uwzględnianie różnic czasowych, w którym występują zachodzące zjawiska. Dane nie muszą trwać tyle samo, łańcuch mogą składać się z różnej liczby znaków z wybranego alfabetu. Dodatkowo bardzo ważną własnością algorytmu jest możliwość porównywania sekwencji, w których brakuje części zebranych danych lub dane wskazują na niedokładności. Pierwotnie metoda została opracowana i służyła do porównywania sygnałów dźwiękowych, lecz ze względu na wysoką skuteczność szybko jej zastosowanie zostało zaadaptowane dla sekwencji opisujących inne zjawiska. Głównym celem algorytmu jest wyznaczenie miary odległości, która służy do klasyfikowania szeregu czasowego. Miarę tę stanowi wyznaczana przez algorytm najmniejsza odle-

głość między dwoma szeregami czasowymi, zwana również optymalną ścieżką. Oznacza ona sytuację, w której badane serie danych różnią się najmniej względem siebie względem zastosowanej miary. Przy dodatkowym założeniu minimalnego minimalnego będącego wyznacznikiem podobieństwa, można klasyfikować wybrane sekwencje jako podobne. Metoda DTW wykonuje swoje działanie korzystając z programowania dynamicznego, a jej złożoność obliczeniowa wynosi O(n*m).

Pierwszym krokiem algorytmu DTW jest wczytanie dwóch szeregów czasowych względem, których obliczana będzie wartość dtw – optymalna odległość pomiędzy dwoma szeregami czasowymi. Na potrzeby dokładnego przedstawienia wszystkich etapów przedstawiony algorytm zostanie omówiony na przykładzie prostych danych numerycznych rysunek 1:

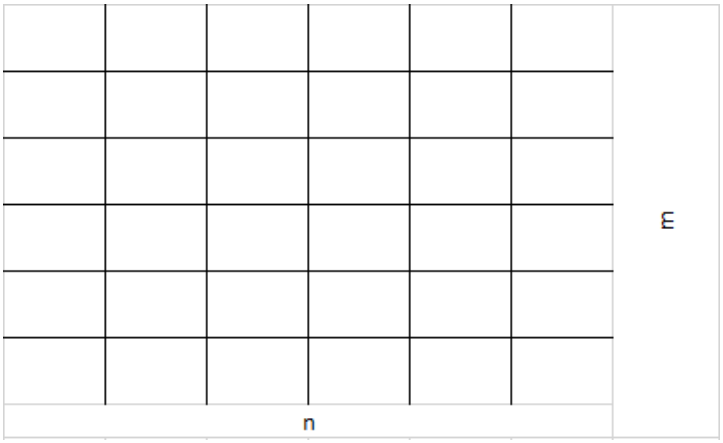
Szereg A = (1, 2, 3, 4, 5, 6)

Szereg B =(2, 4, 6, 8, 10, 12)



Rys. 1. Przykładowe szeregi czasowe A i B

Następnie dla dwóch podanych szeregów należy utworzyć macierz kosztów D o wymiarach n na m gdzie: n oznacza ilość elementów szeregu A, m oznacza ilość elementów szeregu B rysunek 2.



Rys. 2. Pusta macierz kosztów D o wymiarach m×n

Wyliczenie macierzy kosztów D następuje według trzech warunków algorytmu DTW o następującej treści:

1. Warunek początkowy: komórkę macierzy kosztów g[1,1] należy wyliczyć według wzoru: g[1,1] = d(1,1) = |2-1|=1
   gdzie: d(i, j) jest odległością między wartościami we wskazanych punktach ciągu.
2. Warunek wyliczenia pierwszej kolumny i pierwszego wiersza według wzorów:

$$g[1, j] = g[1, j - 1] + d(1, j),$$
$$g[i, 1] = g[i - 1, 1] + d(i, 1)$$

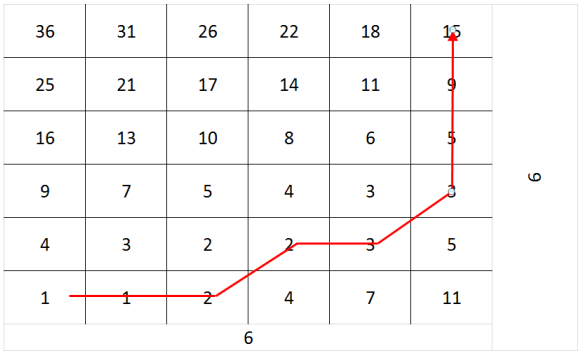
3. Pozostałe komórki macierzy wypełniamy według następującej zależności :
4. $$\mathbf{g[i,j]} = \min \begin{cases} \mathbf{g[i, j-1]} + \mathbf{(di, j)} \\ \mathbf{g[i-1, j-1]} + \mathbf{d(i, j)} \\ \mathbf{g[i-1, j]} + \mathbf{d(i, j)} \end{cases}$$

Po wykonaniu następujących działań utworzona zostanie macierz kosztów D dla szeregów A i B przedstawiona na rysunku 3.

36	31	26	22	18	15	
25	21	17	14	11	9	
16	13	10	8	6	5	
9	7	5	4	3	3	6
4	3	2	2	3	5	
1	1	2	4	7	11	
			6			

Rys. 3. Macierz kosztów D dla omawianego przykładu

Posiadając wypełnioną macierz kosztów, można wyznaczyć optymalną ścieżkę dla porównywanych sekwencji. Wyznaczenie optymalnej ścieżki polega na przejściu przez macierz kosztów rozpoczynając z pozycji D[1,1] i kierując się odpowiednio w górę, po przekątnej lub w prawo wybierając najmniejszą wartość z sąsiedztwa. Wyznaczenie optymalnej ścieżki kończy się na pozycji D[n, m].



Rys. 4. Optymalna ścieżka dokasowaniw (czerwona łamana na macierzy kosztów D)

Optymalna ścieżka dla dwóch przedstawionych zbiorów opisana wzorem

$$\mathbf{d(A,B) = \sum_{i=1}^{6} |m_i - n_i|}$$

wynosi 21. Minimalna odległość dla opisanych zbiorów nazywana również miarą podobieństwa dla tych dwóch zbiorów równa jest DTW(A,B)=15. Możliwe jest również dzielenie macierzy kolumnowo na podmacierze, tak aby można było również znajdować dopasowania częściowe.

4 Implementacja algorytmu w języku C#

Opiszmy implementacje algorytmu DTW przedstawionego w poprzednim rozdziale jako procedury składowanej CLR. Utworzona procedura posłuży do wyszukania w ciągach przechowywanych w relacyjnej bazie danych SQL Server, ciągu o najbardziej zbliżonych cechach do podciągu wejściowego. Podstawą podobieństwa będzie wyznaczona przez algorytm DTW suma odległości pomiędzy porównanymi klatkami wczytanych sekwencji. Implementacja algorytmu została wykonana w języku programowania C#, który jest jednym z dostępnych języków programowania na platformie .NET. Dla lepszego zrozumienia przyjmujemy, że zawarte w bazie danych ciągi danych odzwierciedlają wspomniane wcześniej obiekty lub zjawiska świata rzeczywistego. W tym przypadku dane zgromadzone w bazie danych będą reprezentowały dane wygenerowane przez aplikację, która śledzi gest wykonywany przez ludzki palec na ekranie dotykowym.

Prezentowane rozwiązanie SQL CLR utworzone zostało w projekcie Visual C# SQL CLR Database Project w środowisku programistycznym Microsoft Visual Studio 2010. Przed utworzeniem procedury w projekcie SearchSequence umieszczona została dodatkowa klasa DynamicTimeWarping.cs, która reprezentuje implementację algorytmu DTW. Obiekt tego typu tworzony będzie wewnątrz procedury SQL CLR. Do odzwierciedlenia działania algorytmu DTW wybrany został sposób utworzenia osobnej klasy, a nie tylko metody dla dwóch ciągów, aby przedstawione rozwiązanie umożliwiało łatwe przeniesienie rozwiązania do środowisk innych niż relacyjne bazy danych SQL Server. Cyklem życia obiektów tworzonych wewnątrz procedury zarządzał będzie silnik SQL Engine, co zapewni wysoką wydajność oraz stabilność pracy serwera SQL. Wewnątrz tworzonego kodu wykorzystano instrukcję region poprawiające czytelność kodu. Wewnątrz klasy zdefiniowano następujące zmienne lokalne:

- double[] a; - jednowymiarowa tablica przechowująca reprezentująca ciąg A;
- double[] b; - jednowymiarowa tablica przechowująca reprezentująca ciąg B;
- double[,] D; - macierz kosztów przyjmująca rozmiar zależny od ilości elementów ciągów A i B;
- double DTW; - optymalna odległość między szeregami czasowymi A i B.

W obrębie tej klasy utworzony został konstruktor obiektu DynamicTimeWarping przyjmujący dwa parametry tablicowe typu double. Dane zapisane w tych tablicach posłużą jako ciągi wejściowe

wykorzystane do sprawdzenia podobieństwa metodą DTW. Po wywołaniu konstruktora parametry wejściowe przypisane zostaną odpowiednio do zmiennych globalnych a i b reprezentujących ciągi. Dodatkowo przed pierwszymi obliczeniami wykonane zostanie ustawienie wartości DTW na 0. Kod utworzonego konstruktora klasy wygląda następująco :

```
public DynamicTimeWarping(double[] ciagA,
double[] ciagB)
{a = ciagA;
b = ciagB;
DTW = 0.0;}
```

Pierwszą metodą utworzoną w klasie DynamicTimeWarping jest metoda zwracająca wartość zmiennej DTW typu double.

```
public double getDTW()
{return DTW;}
```

Najważniejszą metodą jest metoda calculateDTW zwracająca zmienną typu void, nazywaną często zerową informacją. W wywołaniu metoda nie przyjmuje żadnych parametrów. W ciele metody wykonują się operacje konieczne do implementacji przedstawionego wcześniej algorytmu DTW. Wewnątrz metody utworzone zostały dodatkowe zmienne typu int. Są to zmienne m i n przechowujące kolejno ilość elementów zbioru A, ilość elementów zbioru B. Zabieg ten został zastosowany ze względu na częstą potrzebę odnoszenia się do tych wartości. Zadeklarowanie dwóch zmiennych typu int i wykonanie jednorazowej operacji policzenia i przypisania do nich ilości elementów zbioru jest wydajniejsze niż każdorazowe odnoszenie się do liczenia elementów zbiorów, które nie będą zmieniane od momentu utworzenia obiektu. Po utworzeniu zmiennych następuje inicjalizacja macierzy kosztów D[,], której rozmiar deklarowany jest w pamięci w oparciu o ilości elementów zbiorów A i B, które zostały zapisane w zmiennych m i n. Następnie według warunku początkowego przypisywana jest wartość pierwszej komórki w macierzy, która znajduje się na pozycji D[0,0]. Stosując warunki wyliczania pierwszego wiersza, a następnie pierwszej kolumny macierzy, a na końcu wzór wypełniania całej macierzy zostaje utworzona pełna macierz kosztów dla porównywanych sekwencji. Ostatnim krokiem metody jest obliczenie minimalnej odległości dla porównywanych zbiorów, która zostaje zapisana w zmiennej dtw. Poniżej przedstawiono pełny kod omawianej metody.

```
public void calculateDTW()
{
//Deklaracja zmiennych wewnętrznych
```

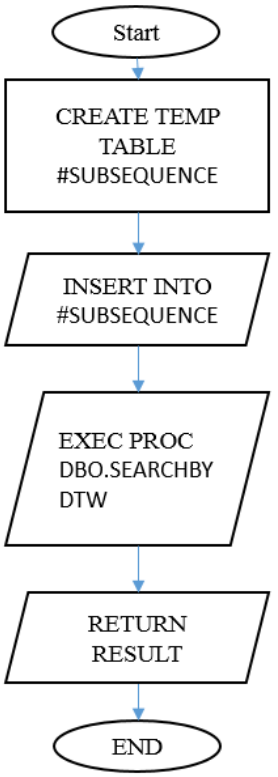
```
int m = a.Length;
int n = b.Length;
D = new double[m, n];
#region Inicjalizacja macierzy kosztów
//Krok 1.Warunek początkowy
D[0, 0] = Math.Abs(a[0] - b[0]);
//Krok 2.Warunek wyliczenia pierwszej kolumny
for (int i = 1; i < m; i++)
{
D[i, 0] = Math.Abs(a[i] - b[0]) + D[i - 1, 0];
}
//Krok 2.Warunek wyliczenia pierwszego wiersza
for (int j = 1; j < n; j++)
{
D[0, j] = Math.Abs(a[0] - b[j]) + D[0, j - 1];
}
#endregion
//Krok 3.Wyliczenie reszty macierzy według wzoru
for (int i = 1; i < m; i++)
{
for (int j = 1; j < n; j++)
{
D[i, j] = Math.Min(D[i, j - 1] + Math.Abs(a[i] -
 b[j]),
Math.Min(D[i-1, j - 1] + Math.Abs(a[i]-b[j]),
D[i - 1, j] + Math.Abs(a[i] - b[j])));
}
}
//Przypisanie wartości optymalnej odległości dla szere
gów a i b
DTW = D[m - 1, n - 1];
}
```

Tak utworzona klasa DynamicTimeWarping.cs posłuży do utworzenia procedury składowanej CLR, za pośrednictwem obiektu pośredniczącego ASSEMBLIES.

```
CREATE ASSEMBLY SearchSequence
AUTHORIZATION [dbo]
FROM 'C:\...\SearchSequence.dll'
WITH PERMISSION_SET = SAFE;
GO
CREATE PROCEDURE SearchByDTW
AS
EXTERNAL NAME
SearchSequence.StoredProcedures.SearchByDTW
```

5 Wyniki testów numerycznych dla serii gestów

Testy zawarte w tym rozdziale zostały wykonane z poziomu narzędzia Microsoft SQL Management Studio. Dane zawarte w bazie zostały wygenerowane automatycznie. Na potrzeby testów przedstawionych w pracy przyjęto, że długość sekwencji nie przekracza 200. Oznacza to, że w bazie znajdują się zbiory zróżnicowanej długości. W bazie umieszczono 1000 ciągów danych, wśród których wyszukiwany będzie ciąg o najbardziej podobnych cechach do podciągu podanego przez użytkownika. Wszystkie pliki testowej bazy danych znajdują się na tym samym dysku co system operacyjny maszyny. Przeprowadzone testy polegają na wyszukaniu sekwencji umieszczonej w zdeklarowanej uprzednio tabeli tymczasowej w zbiorze sekwencji zebranych w relacyjnej bazie danych. Ogólny przebieg wykonania pojedynczego testu przedstawiony został na rysunku 5.



Rys. 5. Ogólny przebieg pojedynczego testu

Poniższy skrypt T-SQL zawiera w sobie przykładowy definiowania dopasowywanego podciągu, który będzie wyszukiwany w bazie.

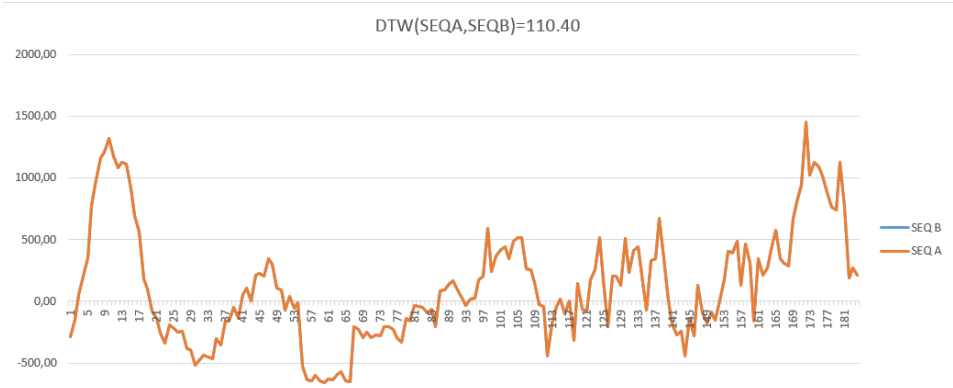
```
DECLARE @BEFORE TIME(7)= SYSDATETIME();
IF object_id('#SUBSEQUENCE') is NULL
DROP TABLE #SUBSEQUENCE;
CREATE TABLE #SUBSEQUENCE
(
ID int IDENTITY(1,1) NOT NULL,
DANE decimal(15,2)
);
INSERT INTO #SUBSEQUENCE VALUES (-5.1), (-2.6), (1),
…(4.8), (3.7);
EXEC [dbo].[SearchByDTW];
DECLARE @AFTER TIME(7)= SYSDATETIME();
SELECT  DATEDIFF(MILLISECOND,@BEFORE,@AFTER) AS
'czas_wyk.(ms)';
```

Dla wykonania powyższego skryptu otrzymujemy następujące wyniki:

ID_OBIE=11, DTW_VALUE=70.71, czas_wyk=4973 (ms).

Wynik należy interpretować w ten sposób, że najbardziej zbliżonym ciągiem do wyszukiwanego podciągu jest zawarty w bazie ciąg o numerze 11. Wskaźnik dopasowania dtw dla podciągu podanego przez użytkownika oraz wskazanego jako najbardziej podobny w bazie wynosi 70.71. W zależności od przyjętego progu minimalnego i otrzymanego wyniku możemy przyjąć, że wyszukiwany gest został rozpoznany. Porównanie z 1000 rekordów , czyli wykonanie algorytmu DTW pomiędzy wskazanym podciągiem i tysiącem szeregów czasowych zawartych w bazie oscyluje w granicach 5 sekund. Dodatkowe testy polegające na wykonaniu w pętli o tysiącu powtórzeniach wyżej wymienionego skryptu dla tysiąca różnych podciągów wejściowych, w przeliczeniu na pojedynczy test, również wynosi około 5 sekund. Przyglądając się zachowaniu silnika bazodanowego za pomocą menedżera zadań wbudowanego w system operacyjny, można zauważyć następujące wnioski. Użycie opisanego skryptu nie angażuje dużej ilości pamięci RAM, lecz korzysta z zaalokowanych już przez silnik SQL Engine zasobów RAM. Głównymi podzespołami, które wpływają na szybkość wykonania procedury CLR jest procesor oraz pamięć RAM, których użycie dla opisanej instancji widać w menedżerze zadań.

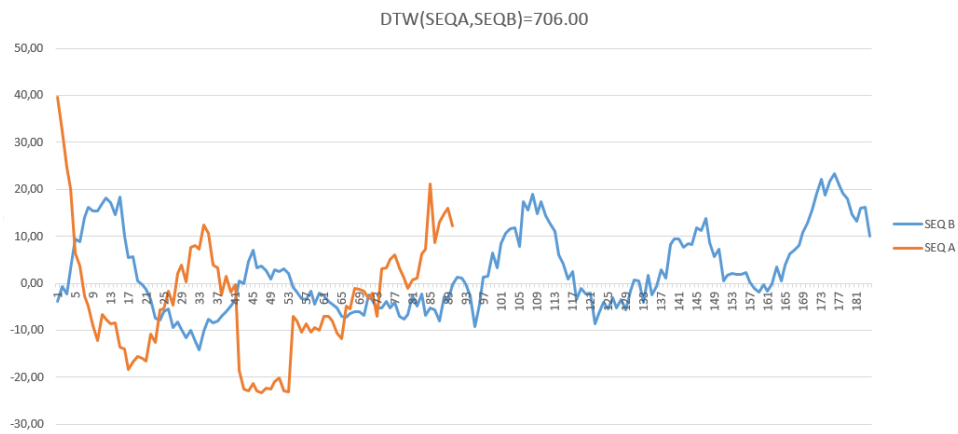
Poza testami wydajności przedstawionymi powyżej wykonano szereg innych prób mających na celu weryfikację jakości zastosowanych rozwiązań. Poniżej przedstawiono dwa spośród nich. W pierwszym wprowadzone zostały dwa łańcuccy o równiej dość dużej długości oraz o bardzo niewiele różniących się wartościach. Różnice nie przekraczały 1% wartości co powoduje, że nie widoczne są one na prezentacji graficznej rysunek 6 co można sprawdzić analizując rzeczywiste wartości rysunek 7.



Rys. 6. Prezentacja graficzna porównywanych łańcuchów

Elementy obiektu	Ilość elementów	Dopasowany ciąg (ID_OBIE)	Ilość elementów dopasowanego ciągu	DTW
-286.02, -145.52, 56.80, 202.92, 354.66, 776.16, 984.10, 1163.94, 1208.90, 1321.30, 1175.18, 1085.26, 1124.60, 1113.36, 911.04, 691.86, 562.60, 180.44, 101.76, -78.08, -139.90, -257.92, -342.22, -190.48, -218.58, -246.68, -241.06, -381.56, -392.80, -516.44, -471.48, -437.76, -449.00, -465.86, -302.88, -353.46, -162.38, -156.76, -49.98, -134.28, 56.80, 107.38, 6.22, 214.16, 225.40, 202.92, 349.04, 298.46, 107.38, 90.52, -72.46, 39.94, -55.60, -10.64, -522.06, -634.46, -645.70, -600.74, -645.70, -656.94, -628.84, -634.46, -589.50, -567.02, -645.70, -651.32, -201.72, -229.82, -291.64, -246.68, -291.64, -269.16, -280.40, -201.72, -201.72, -229.82, -297.26, -330.98, -139.90, -156.76, -33.12, -38.74, -49.98, -94.94, -61.22, -201.72, 84.90, 90.52, 141.10, 169.20, 90.52, 28.70, -33.12, 17.46, 28.70, 174.82, 202.92, 590.70, 242.26, 365.90, 416.48, 444.58, 343.42, 489.54, 517.64, 517.64, 264.74, 259.12, 152.34, -27.50, -38.74, -443.38, -179.24, -38.74, 17.46, -100.56, 0.60, -314.12, 146.72, -61.22, -78.08, 174.82, 259.12, 517.64, 174.82, -207.34, 202.92, 202.92, 129.86, 506.40, 236.64, 410.86, 444.58, 152.34, -72.46, 332.18, 343.42, 675.00, 343.42, 6.22, -184.86, -274.78, -241.06, 443.38, -139.90, -280.40, 129.86, -89.32, -173.62, -89.32, -151.14, 11.84, 180.44, 405.24, 394.00, 489.54, 129.86, 467.06, 309.70, -162.38, 343.42, 208.54, 270.36, 422.10, 573.84, 349.04, 309.70, 287.22, 669.38, 809.88, 944.76, 1450.56, 1023.44, 1124.60, 1090.88, 1017.82, 882.94, 759.30, 742.44, 1124.60, 764.92, 191.68, 270.36, 208.54,	184	-286.62, -146.12, 56.20, 202.32, 354.06, 775.56, 983.50, 1163.34, 1208.30, 1320.70, 1174.58, 1084.66, 1124.00, 1112.76, 910.44, 691.26, 562.00, 179.84, 101.16, -78.68, -140.50, -258.52, -342.82, -191.08, -219.18, -247.28, -241.66, -382.16, -393.40, -517.04, -472.08, -438.36, -449.60, -466.46, -303.48, -354.06, -162.98, -157.36, -50.58, -134.88, 56.20, 106.78, 5.62, 213.56, 224.80, 202.32, 348.44, 297.86, 106.78, 89.92, -73.06, 39.34, -56.20, -11.24, -522.66, -635.06, -646.30, -601.34, -646.30, -657.54, -629.44, -635.06, -590.10, -567.62, -646.30, -651.92, -202.32, -230.42, -292.24, -247.28, -292.24, -269.76, -281.00, -202.32, -202.32, -230.42, -297.86, -331.58, -140.50, -157.36, -33.72, -39.34, -50.58, -95.54, -61.82, -202.32, 84.30, 89.92, 140.50, 168.60, 89.92, 28.10, -33.72, 16.86, 28.10, 174.22, 202.32, 590.10, 241.66, 365.30, 415.88, 443.98, 342.82, 488.94, 517.04, 517.04, 264.14, 258.52, 151.74, -28.10, -39.34, -443.98, -179.84, -39.34, 16.86, -101.16, 0.00, -314.72, 146.12, -61.82, -78.68, 174.22, 258.52, 517.04, 174.22, -207.94, 202.32, 202.32, 129.26, 505.80, 236.04, 410.26, 443.98, 151.74, -73.06, 331.58, 342.82, 674.40, 342.82, 5.62, -185.46, -275.38, -241.66, 443.98, -140.50, -281.00, 129.26, -89.92, -174.22, -89.92, -151.74, 11.24, 179.84, 404.64, 393.40, 488.94, 129.26, 466.46, 309.10, -162.98, 342.82, 207.94, 269.76, 421.50, 573.24, 348.44, 309.10, 286.62, 668.78, 809.28, 944.16, 1449.96, 1022.84, 1124.00, 1090.28, 1017.22, 882.34, 758.70, 741.84, 1124.00, 764.32, 191.08, 269.76, 207.94	184	110.40

Rys. 7. Numeryczna reprezentacja porównywanych łańcuchów oraz wynik porównania



Rys. 8. Prezentacja graficzna porównywanych łańcuchów

Elementy obiektu	Ilość elementów	Dopasowany ciąg (ID_OBIE)	Ilość elementów dopasowanego ciągu	DTW
39.70, 32.50, 24.70, 20.10, 6.50, 3.70, -2.70, -4.90, -9.10, -12.10, -6.70, -7.70, -8.70, -8.50, -13.50, -13.90, -18.30, -16.70, -15.50, -15.90, -16.50, -10.70, -12.50, -5.70, -5.50, -1.70, -4.70, 2.10, 3.90, 0.30, 7.70, 8.10, 7.30, 12.50, 10.70, 3.90, 3.30, -2.50, 1.50, -1.90, -0.30, -18.50, -22.50, -22.90, -21.30, -22.90, -23.30, -22.30, -22.50, -20.90, -20.10, -22.90, -23.10, -7.10, -8.10, -10.30, -8.70, -10.30, -9.50, -9.90, -7.10, -7.10, -8.10, -10.50, -11.70, -4.90, -5.50, -1.10, -1.30, -1.70, -3.30, -2.10, -7.10, 3.10, 3.30, 5.10, 6.10, 3.30, 1.10, -1.10, 0.70, 1.10, 6.30, 7.30, 21.10, 8.70, 13.10, 14.90, 15.90, 12.30	90	-3.90, -0.70, -2.30, 3.10, 9.50, 8.90, 14.00, 16.20, 15.30, 15.40, 16.90, 18.20, 17.10, 14.70, 18.30, 10.00, 5.50, 5.70, 0.60, -0.20, -1.20, -3.50, -7.50, -7.80, -6.00, -5.50, -9.50, -8.30, -9.80, -11.50, -10.00, -12.30, -14.20, -10.10, -7.60, -8.50, -8.10, -7.00, -6.10, -4.80, -3.50, 0.60, -0.10, 4.60, 7.00, 3.20, 3.60, 2.70, 0.90, 2.90, 2.60, 3.00, 2.20, -0.80, -1.90, -3.20, -3.40, -1.60, -4.40, -2.00, -2.70, -3.90, -4.40, -5.30, -7.00, -7.20, -6.50, -6.00, -6.00, -6.80, -2.40, -3.60, -5.20, -5.30, -3.90, -5.20, -4.10, -7.00, -7.70, -6.60, -2.60, -4.80, -2.30, -6.80, -5.20, -5.60, -8.10, -2.90, -3.90, -0.20, 1.40, 1.10, -0.10, -2.70, -9.30, -4.40, 1.40, 1.60, 6.50, 3.30, 8.50, 10.70, 11.60, 11.90, 7.80, 17.30, 15.50, 18.90, 14.80, 17.40, 14.40, 12.60, 11.10, 6.00, 4.10, 0.90, 2.50, -3.40, -1.00, -2.20, -2.10, -8.70, -6.40, -4.10, -5.40, -3.00, -5.20, -3.40, -5.60, -1.60, 0.70, 0.50, -4.10, 1.70, -2.40, -0.70, 2.90, 1.20, 8.20, 9.40, 9.40, 7.70, 8.50, 8.30, 11.80, 11.20, 13.90, 8.70, 5.70, 7.20, 0.60, 1.80, 2.20, 1.90, 2.00, 2.30, 0.30, -1.10, -1.90, -0.30, -1.70, 0.00, 3.40, 0.60, 3.80, 6.30, 7.10, 8.10, 10.80, 12.90, 15.50, 19.20, 22.20, 18.70, 21.70, 23.30, 21.00, 19.20, 18.00, 14.60, 13.30, 15.90, 16.20, 10.10,	184	706.00

Rys. 9. Numeryczna reprezentacja porównywanych łańcuchów oraz wynik porównania

Z kolei na rysunkach 8 i 9 przedstawione zostały łańcuchy o różnych długościach oraz o zdecydowanie mniej podobnych przebiegach. Rysunek 9 zawiera poza reprezentacją numeryczną porównywanych danych wynik działania algorytmu. Tego typu próby potwierdzają skuteczność zaproponowanej metody i jej realizacji programistycznej dla danych o bardzo zróżnicowanej postaci.

6 Podsumowanie

W niniejszej pracy zaprezentowane zostało tworzenie procedur użytkownika z użyciem technologii CLR dla serwera bazy danych Microsoft SQL Server. Mechanizm ten zastosowany został do realizacji praktycznej algorytmu mapowania, dopasowania łańcuchów nad nieskończonym alfabetem z zastosowaniem metody DTW (Dynamic Time Warping). Utworzona w części praktycznej procedura składowa CLR znakomicie spisuje się podczas testowania, a jej niskie zapotrzebowanie na zasoby systemu operacyjnego i wysoka prędkość wykonania kwalifikuje ją do adaptacji na systemach produkcyjnych. Co więcej używanie procedury w środowisku produkcyjnym, nieporównywalnie lepszym pod względem parametrów od maszyny, na której zrealizowane zostały cele testowe, przyniosłoby jeszcze lepsze rezultaty wydajnościowe.

Ze względu na bardzo duże zapotrzebowanie na realizację praktyczną przeszukiwania nieliniowe (DTW), procedura stanowić może przedmiot dalszych badań nad rozwojem jej efektywności w celu poprawienia wydajności. W prosty sposób można w nim zredukować powtarzające się przebiegi petli, co za tym idzie zmniejszyć ilość wykonywanych operacji i wpłynąć pozytywnie na wydajność prezentowanej procedury CLR. W pracy ten zabieg nie został celowo

zastosowany, aby ukazać dokładne odzwierciedlenie klasycznej postaci algorytmu.

Dodatkowo zaproponowano tylko porównywanie ciągów skalarnych. Możliwa jest dalszy rozwój metody w kierunku uwzględnienia alfabetów wektorowych czyli takich gdzie jednym elementem łańcucha jest układ współrzędnych określających położenie punktu w przestrzeni n wymiarowej. Jedyna znacząca modyfikacja polegałaby wówczas na zmianie sposobu określenia odległości między elementami szeregu korzystając z jednej z kilku możliwych metryk [13], [14]. Najbardziej ogólną metryką jest odległość Minkowskiego definiowana, jako

$$d_n(x_i, x_j) = \left(\sum_{k=1}^{d} (|x_{i,k} - x_{j,k}|)^n \right)^{\frac{1}{n}}$$

którą możemy sprowadzić do popularnych przypadków szczegółowych, odległości miejskiej (Manhattan) (n=1)

$$d_1(x_i, x_{ij}) = \sum_{k=1}^{d} |x_{i,k} - x_{j,k}|$$

odległości Euklidesowej (n=2)

$$d_2(x_i, x_j) = \left(\sum_{k=1}^{d} (x_{i,k} - x_{j,k})^2 \right)^{\frac{1}{2}}$$

odległości Czebyszewa (n→∞)

$$d_\infty(x_i, x_j) = \lim_{n \to \infty} \left(\sum_{k=1}^{d} |x_{i,k} - x_{j,k}|^n \right)^{\frac{1}{n}}$$

Możliwe jest również wykorzystanie miary kosinusowej danej przez zależność [5]

$$d(x, y) = 1 - \frac{x^{\circ} y}{|x| \cdot |y|}$$

Dalszy rozwój prac nad przedstawioną problematyką w kontekście stosowania bardziej złożonych elementów łańcucha oraz stosowania i oceny jakości stosowanej metryki, a także zastosowania porównania przybliżonego lub rozmytego jest przewidziany w dalszych pracach.

7 Literatura

[1] Chomsky N., *Three models for the description of language*, IRE Transactions on Information Theory, 1956

[2] Brown P., Della Pietra V., Della Pietra S., Mercer R., *The mathematics of statistical machine translation: parameter estimation.* Journal Computational Linguistics – Special issue on using large corpora: II archive Volume: 19 Issue: 2, pp. 263-311, 1993.

[3] Jaworski R., *Computing transfer score in Example-Based Machine Translation*, Lecture Notes in Computer Science 6008 (LNCS), Springer -Verlag, pp. 406-416, 2010

[4] Salak A., Pelikant A., *Algorithms of writing music notation in database,* XIII INTERNATIONAL CONFERENCE SYSTEM MODELLING and CONTROL, Zakopane 12-14 oct. 2009

[5] Cieślewicz J., Pelikant A., *Zastosowanie sieci językowych w reprezentacji dokumentów tekstowych*, Studia Informatica, Volume 32, No 2A (96) Silesian University of Technology Press, Gliwice 2011, pp. 517-526.

[6] Lidke M., Pelikant A., *Automatyczne wykrywanie słów kluczowych*, Zeszyty Naukowe Wyższej Szkoły Informatyki w Łodzi, Vol. 11, Nr 3, 2012 ss. 25-50

[7] Frikken Keith B., *Practical Private DNA String Searching and Matching through Efficient Oblivious Automata Evaluation*, Data and Applications Security XXIII, Lecture Notes in Computer Science Volume 5645, 2009, pp 81-94

[8] Blanton M., Aliasgari M., *Secure Outsourcing of DNA Searching via Finite Automata*, Data and Applications Security and Privacy XXIV, Lecture Notes in Computer Science Volume 6166, 2010, pp 49-64

[9] Chen Qing K., Hertz Gerald Z., Stormo G. D., *MATRIX SEARCH 1.0: a computer program that scans DNA sequences for transcriptional elements using a database of weight matrices*, Oxford Journals, Life Sciences & Mathematics & Physical Sciences, Bioinformatics, Volume 11, Issue 5, ss. 563-566.

[10] Mrozek D., Malysiak B., *Searching for strong structural protein similarities with EAST*, Computer Assisted Mechanics and Engineering Sciences 14 (4), s. 681, 2007

[11] Małysiak-Mrozek B., Momot A., Mrozek D., Momot M., *Architektura hierarchicznego systemu wieloagentowego dla procesu poszukiwania podobieństwa białek*, Studia Informatica 33 (2A), ss. 83-97, 2012

[12] Małysiak-Mrozek B., Kozielski S., Mrozek D., *Server-Side Query Language for Protein Structure Similarity Searching*, Human–Computer Systems Interaction: Backgrounds and Applications 2, ss. 395-415, Springer 2012

[13] Niewiadomy D., Pelikant A., Query by Voice Example and sound similarity based on the Dynamic Time Warping algorithm, Electrical Review 2010-8

[14] D. Niewiadomy, A. Pelikant – *Query by Voice Example and sound similarity based on the DynamicTime Warping algorithm*, XIII INTERNATIONAL CONFERENCE SYSTEM MODELLING and CONTROL, Zakopane 12-14 oct. 2009

[15] Konopka E., Pelikant A., *Funkcje i typy użytkownika CLR w zadaniach statystycznych*, Zeszyty Naukowe Wyższej Szkoły Informatyki w Łodzi, Vol. 11, Nr 2, 2012 ss. 5-30

[16] Pilny P., Pelikant A., *Typy użytkownika CLR – wprowadzenie obiektowości do relacyjnej bazy danych*, Zeszyty Naukowe Wyższej Szkoły Informatyki w Łodzi, Vol. 11, Nr 2, 2012 ss. 51-81

[17] Pelikant A., *Zastosowanie typów obiektowych w przetwarzaniu analitycznym,* Roczniki Kolegium Analiz Ekonomicznych, zeszyt 25/2012, SGH 2012, ISSN 1232-4671 pp. 231-250

SEARCHING FOR SUBSTRING IN STRING USING CLR OBJECTS

Summary – The purpose of this work is to present the possibility to use mapped to the procedural elements Transact SQL CLR object classes that are created on the .NET platform in a complex processing algorithms. There are presented theoretical algorithms for matching chains for finite symbol set alphabets. For introduced infinite symbol set alphabets solutions may not be easily modified, so it was proposed the algorithm DTW (Dynamic Time Warping), which was programmed using the mapping rules for procedural extension to SQL. There where shown elements of the practical implementation and the experimental results of matching gestures were discuss.

Keywords: pattern matching, dynamic time warping, DTW, CLR programming