Laboratorium
Komputerowych Systemów Rozpoznawania
Projekt 2. Lingwistyczne podsumowania baz danych

Robert Nowotniak, 120308
Michał Wysokiński, 120404

Data oddania projektu: 10 grudnia 2007

Spis treści

Spis rysunków

1 Opis projektu

Celem projektu było napisanie narzędzia służącego do tworzenia lingwistycznych podsumowań baz danych.

Program został napisany z zamysłem, by był on „sterowany danymi" (ang. *data-driven*) — dostosowanie do konkretnej bazy danych, do konkretnej dziedziny problemu odbywa się całkowicie za pomocą struktur danych, a nie jest osadzone w kodzie programu. Dzięki temu program pozwala na **analizowanie dowolnych baz danych**.

Program umożliwia generowanie podsumowań lingwistycznych w I i II formie kanonicznej.

Przy projektowaniu pakietu do obliczeń rozmytych możliwe były dwa podejścia... – reprezentacja ciągłych przestrzeni rozważań za pomocą jej dyskretnych podziałów (z pewnych niewielkim krokiem) oraz reprezentacja funkcji przynależności w postaci analitycznej. Do zastosowań niniejszego zadania podjęliśmy decyzję o wyborze drugiej z tych metod.

Program został napisany w języku **C#** i środowisku **Visual Studio 2005**. Komunikacja z bazą danych odbywa się za pomocą sterownika **ODBC** — w ten sposób zostało uzyskane uniezależnienie projektu od konkretnego silnika RDBMS. Testową analizowaną bazą była baza na **PostgreSQL**, będąca pełną bazą danych **Klubu Sportowego**.

Testowanie stworzonej biblioteki obliczeń rozmytych przeprowadzone zostało za pomocą narzędzia **NUnit** oraz napisanego zestawu testów jednostkowych do sprawdzania poprawności działania biblioteki. Przedstawia to rysunek 23.

2 Możliwości programu

Program umożliwia użytkownikowi:

1. Generowanie podsumowań typu I
2. Generowanie podsumowań typu I dla koniunkcji wielu sumaryzatorów
3. Generowanie podsumowań typu II (kwalifikacja)
4. Generowanie podsumowań typu II (kwalifikacja) dla wielu sumaryzatorów
5. Analizowanie **dowolnej** bazy danych (via ODBC) – rys. 7
6. Tworzenie nowych podmiotów, kwantyfikatorów, cech, wartości, funkcji przynależności
7. Zapis i odczyt całej konfiguracji

Program umożliwia zapis do pliku całego drzewa obiektów generatora — czyli całej hierarchii podmiotów, kwantyfikatorów, własności obiektów oraz ich zbiorów rozmytych opisujących ich cechy. Zapis wykonywany jest serializatora binarnego (BinaryFormatter. Bardziej odpowiadającym nam formatem serializacji byłby format pliku XML – jednak podstawowe mechanizmy serializacji środowiska .NET 2.0 okazały się niewystarczające, by tego serializację tego typu dla złożonego drzewa obiektów zrealizować w prosty sposób.

Jeśli użytkownik wybierze jakiekolwiek kwalifikatory, to samoczynnie oznacza to że generowane będą podsumowania w II formie.

Wybór wielu sumaryzatorów oznacza żądanie generowania podsumowań z ich koniunkcjami.

Ponieważ program pozwala na połączenie się z dowolną bazą danych w celu jej analizy, wpływa to w pewnym stopniu na szybkość jego działania. Ponieważ nie jest on uzależniony od konkretnej budowy tabel, musi w dynamiczny sposób rozpoznawać typy tabel.

Zauważyliśmy, że – w szczególności dla języka polskiego! – stosowanie modyfikatorów (ang. *hedges*) nie daje zbyt dobrego rezultatu. Dodawanie modyfikatorów zwiększa, proporcjonalnie, liczbę generowanych podsumowań, spośród których zdecydowana większość jest bardzo niepoprawka gramatycznie. Zamiast tego, naszym zdaniem, lepiej jest po prostu dodawać nowe kwantyfikatory.

Użytkownik, który za pomocą stworzonego programu chce dokonać analizy dla nowej bazy danych, musi postępować wg następującej procedury.

1. Ustawienie parametrów połączenia z bazą — za pomocą konfiguratora
2. Utworzenie zbioru podmiotów podsumowań, na podstawie tabel, istniejących w bazie
   Przypisanie podmiotom tabel, do których się odnoszą.
3. Zdefiniowanie kwantyfikatorów lingwistycznych — wraz ze zdefiniowaniem ich funkcji przynależności.
4. Przypisanie kwantyfikatorów do podmiotów. Kwantyfikatory oznaczone jako „globalne” odnoszą się do wszystkich podmiotów
5. Utworzenie cech, które opisują utworzone przez użytkownika podmioty podsumowań. Przypisanie cechom kolumn, należących do tabel poszczególnych podmiotów oraz zdefiniowanie wartości zmiennych lingwistycznych opisujących cechy obiektów.
6. Zapisanie całej powyższej konfiguracji do pliku

Po powyższej konfiguracji generatora podsumowań, przykładowy przebieg generowania podsumowań lingwistycznych bazy może przebiegać wg następującego scenariusza:

1. Użytkownik wybiera analizowany podzbiór obiektów danego podmiotu, za pomocą kwalifikatorów.

2. Użytkownik wybiera zbiór wartości cech, które podlegać będą analizie

Do celów niniejszego projektu stworzyliśmy dwie dodatkowe kontrolki Windows Forms: FeaturesSelectionPanel oraz **FunctionPlot**. Pierwsza z nich pozwala na reprezentację oraz wybór cech i ich wartości – dla przypisanego jej podmiotu. Druga z kontrolek (FunctionPlot) służy do rysowania wykresu dowolnych funkcji przynależności (MembershipFunction).

3 Symboliczna reprezentacja postaci funkcji przynależności

W stworzonej biblioteki obliczeń rozmytych użyliśmy **symbolicznej reprezentacji** funkcji przynależności. Moduł został stworzony w paradygmacie programowania obiektowego, podstawową klasą pakietu jest klasa abstrakcyjna *MembershipFunction*.

Obiekt tej klasy stanowi korzeń **abstrakcyjnego drzewa składniowego** funkcji przynależności.

Podstawowe postaci funkcji przynależności (trapezoidalna, gussowska, singletonowa) należy tu traktować jako **wyrażenia terminalne**. Natomiast funkcje *Complement*-, *Union*- i *IntersectionMembershipFunction* — jako **obiekty-funktory**, które jednocześnie egzemplifikują abstrakcyjny typ bazowy *MembershipFunction*, realizując jego interfejs, oraz posiadają odpowiednią liczbę dowolnych funkcji przynależności, będących ich argumentami[1].

Podstawowe funkcje przynależności (wyrażenia terminalne) zdefiniowane w programie to:

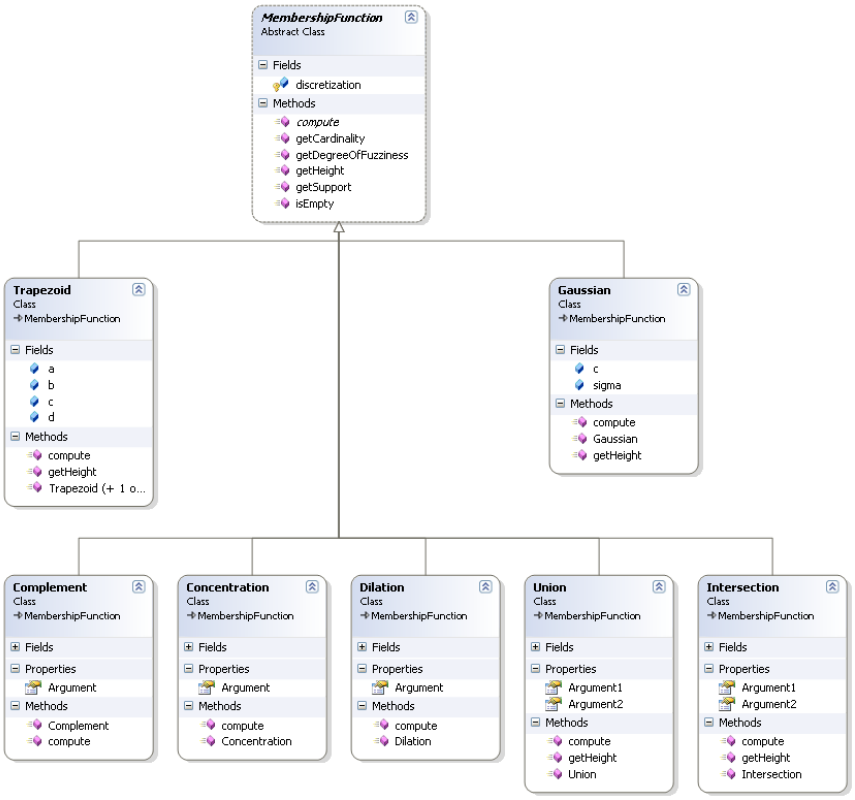
1. Trapezoidalna (określana za pomocą czterech parametrów a, b, c, d). Zauważmy, że funkcja trapezoidalna jednocześnie realizuje od razu funkcje: singletonową, Trójkątną i prostokątna

2. Gaussowska — o parametrach: c oraz σ.

Dla przykładu, klasa fuzzy.MembershipFunctions.Complement (dziedzicząca z klasy abstrakcyjnej *MembershipFunction*) nadpisuje **wirtualną metodę compute**, wykorzystując w elegancki sposób argument „funkcji" Complement.

Definicja nadpisanej metody wirtualnej:

[1]Ciekawym dodatkowym rozszerzeniem, które można byłoby zaimplementować, byłoby napisanie parsera który budowałby drzewo składniowe dla funkcji przynależności podawanych przez użytkownika w postaci łańcucha tekstowego np. `gauss(0.5,1)+dilation(trapezoid(0.1,0.3,0.9))`



Rysunek 1: Diagram klas funkcji przynależności (*MembershipFunction*)

```
public override float compute(object o)
{
   return 1 - argument.compute(0);
}
```

Podobnie zaimplementowana jest metoda wirtualna w klasie *Union*, gdzie liczona jest suma funkcji przynależności.

```
public override float compute(object o)
{
   return Math.Max(argument1.compute(o), argument2.compute(o));
}
```

Abstrakcyjna klasa *MembershipFunction* zawiera ogólne implementacje operacji takich jak: znajdowanie α-przekroju, sprawdzenie wypukłości. Klasy konkretne, dziedziczące z *MembershipFunction* mogą nadpisywać jej metody wirtualne (np. isConvex()), dostarczając „lepsze" implementacje, opierając się na własnościach danej funkcji przynależności.

Przykładowo, implementacja funkcji obliczającej wysokość w MembershipFunction wykorzystuje iteracyjne przechodzenie przez cały zakres dzie-

dziny – z pewnym małym krokiem – w celu znalezienia wartości maksymalnej. Jednak np. implementacja tej metody w klasie dziedziczącej – Union, czyli funkcja będąca sumą funkcji przynależności – wykorzystuje własność, że wysokość sumy jest sumą (s-normą) wysokości jej argumentów.

Podstawową klasą biblioteki obliczeń rozmytych jest klasa **FuzzySet**, posiadająca obiekt realizujący interfejs *MembershipFunction*.

Klasa FuzzySet dostarcza następujące wygodne konstruktory (wystarczające do podstawowych „zastosowań" zbiorów):

Utworzenie zbioru rozmytego w gęstej przestrzeni, o singletonowej funkcji przynależności i wartości 1 tylko dla argumentu x:

```
FuzzySet(float x)
```

Jak wyżej, ale z funkcją trapezoidalną:

```
FuzzySet(float a, b, c, d)
```

Zbiór rozmyty w dziedzinie dyskretnej, z funkcją przyjmującą wartość x tylko dla argumentu o etykiecie `label`:

```
FuzzySet(string label, float x)
```

4 Kluczowe diagramy klas

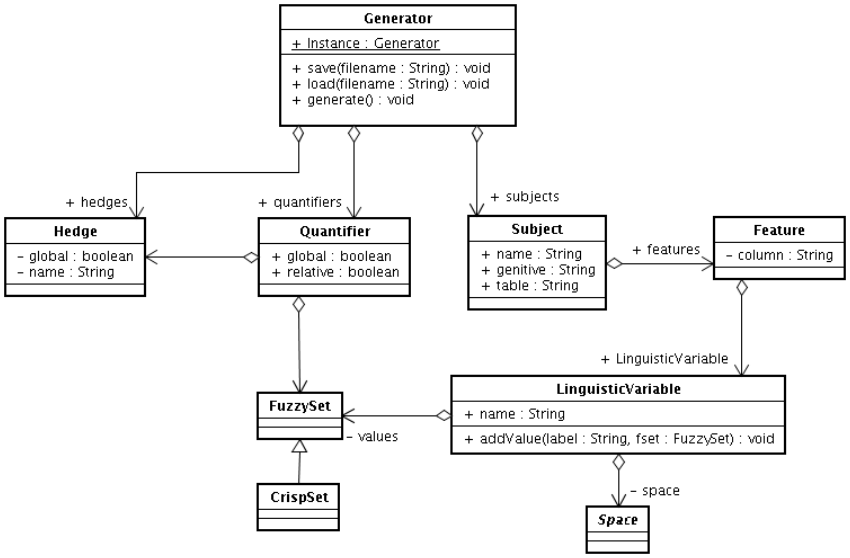
Diagramy kluczowych klas projektu przedstawiają rysunki 3, 4 i 2.

Klasy z pakietu LinguisticSummary.lingsumm – odpowiedzialne za tworzenie podsumowań – przedstawione są na rysunku 2.

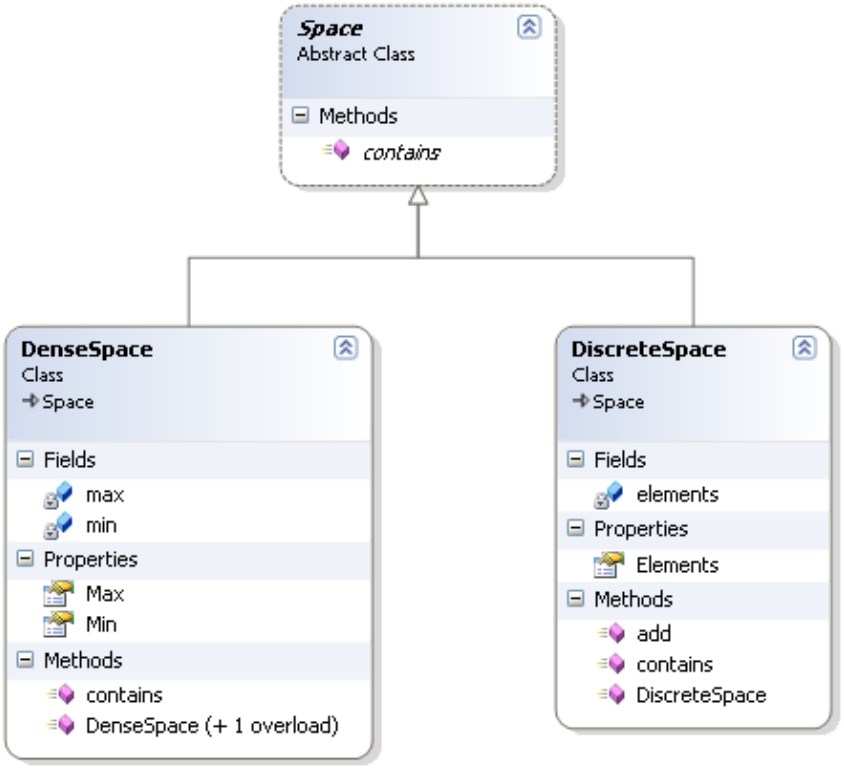
Zamieszczone diagramy zostały znacznie uproszczone, w celu zachowania ich przejrzystości.

Przestrzeń rozważań (obiekt posiadany m.in. przez zmienne lingwistycznej *LinguisticVariable*) reprezentowana jest przez klasę abstrakcyjną *Space*, którą rozszerzają klasy konkretne Discrete- i DenseSpace.

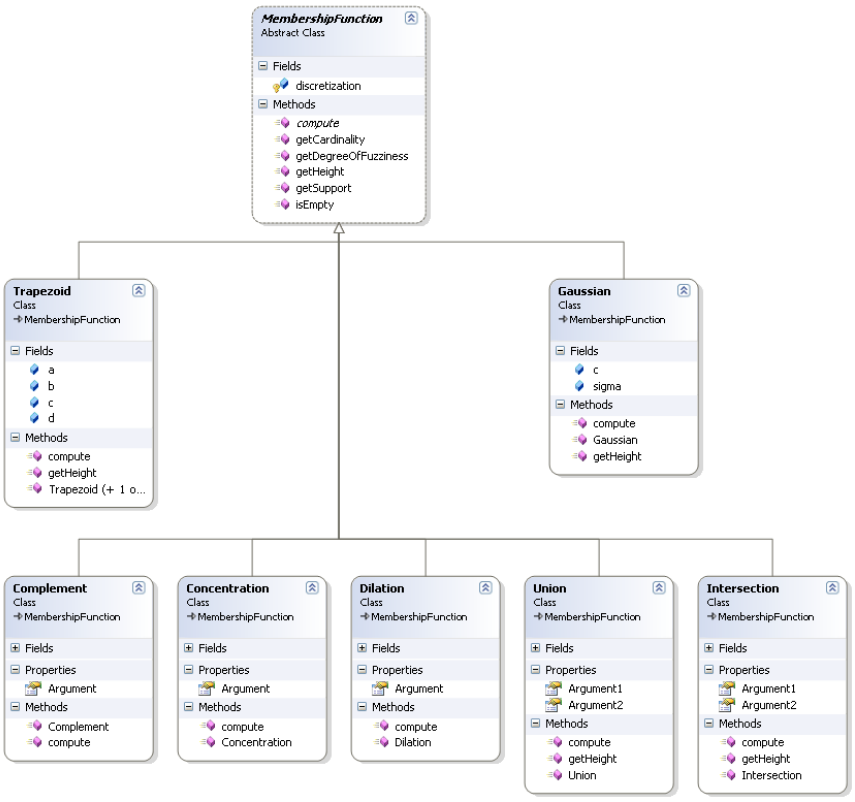
Klasy z pakietu LinguisticSummary.fuzzy.MembershipFunction przedstawia diagram 4 — jest to diagram klas, które służą do budowy **abstrakcyjnego drzewa składniowego**.



Rysunek 2: Diagram klas generatora podsumowań



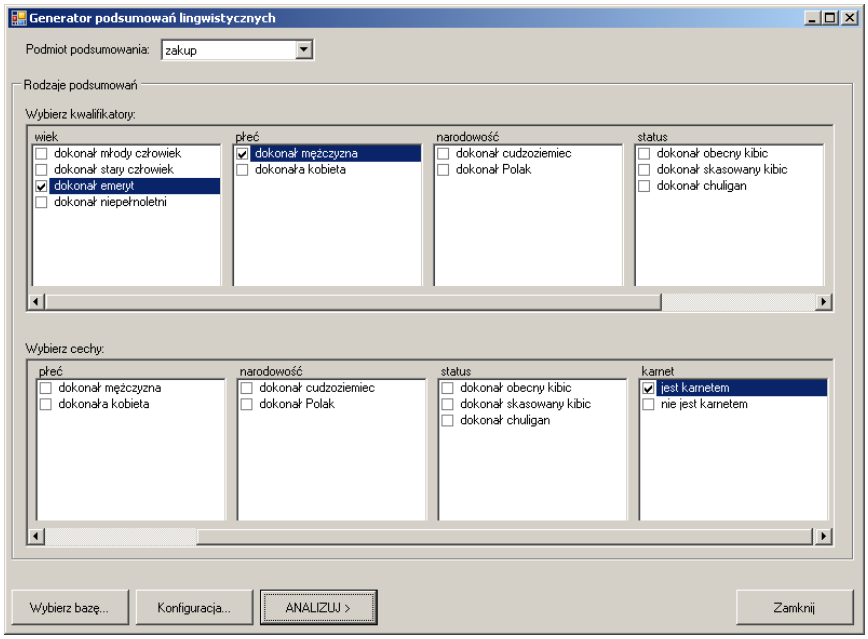
Rysunek 3: Diagram klas przestrzeni rozważań



Rysunek 4: Diagram klas funkcji przynależności (*MembershipFunction*)

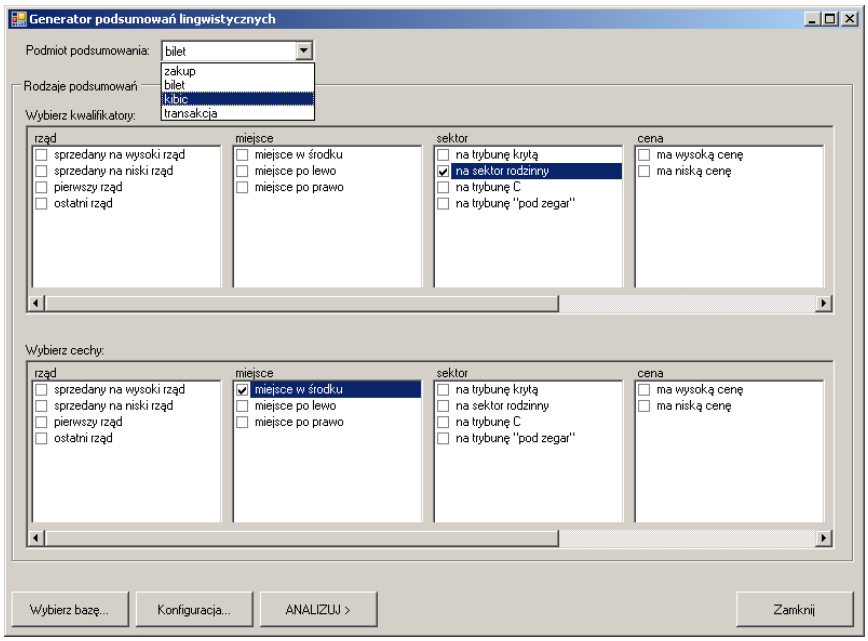
5 Interfejs użytkownika (moduł podstawowy)

Główne okno programu ma bardzo prostą budowę (w odróżnieniu od okna konfiguracyjnego). Lista rozwijana pozwala na wybór podmiotu, dla którego będzie generowane podsumowanie.

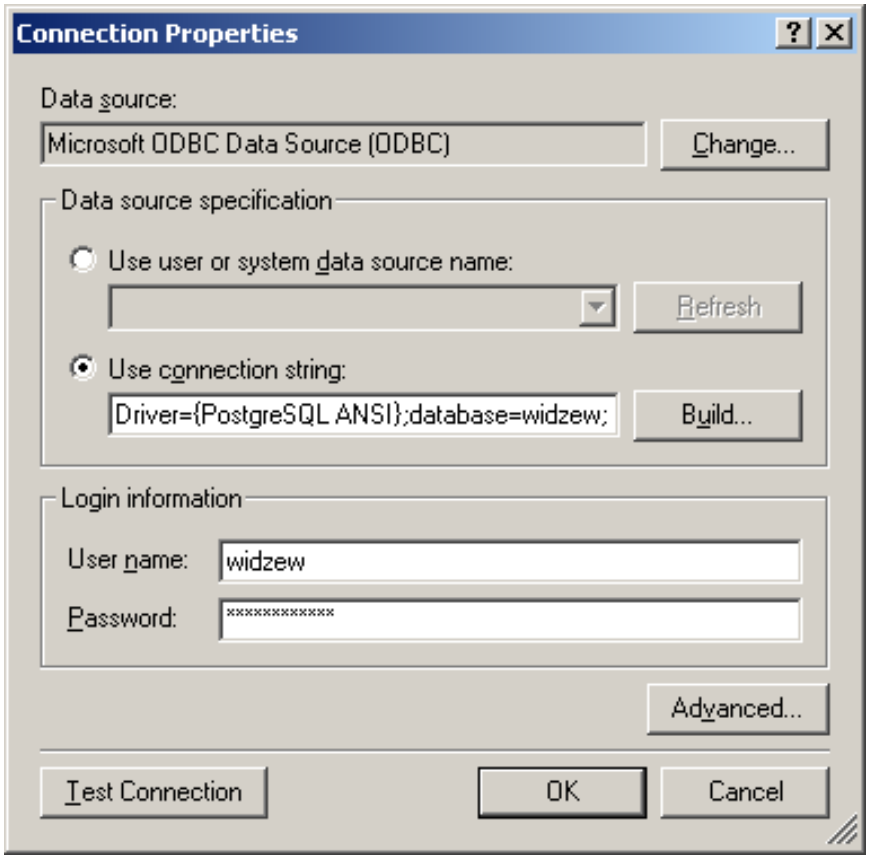


Rysunek 5: Główne okno generatora (moduł podstawowy)

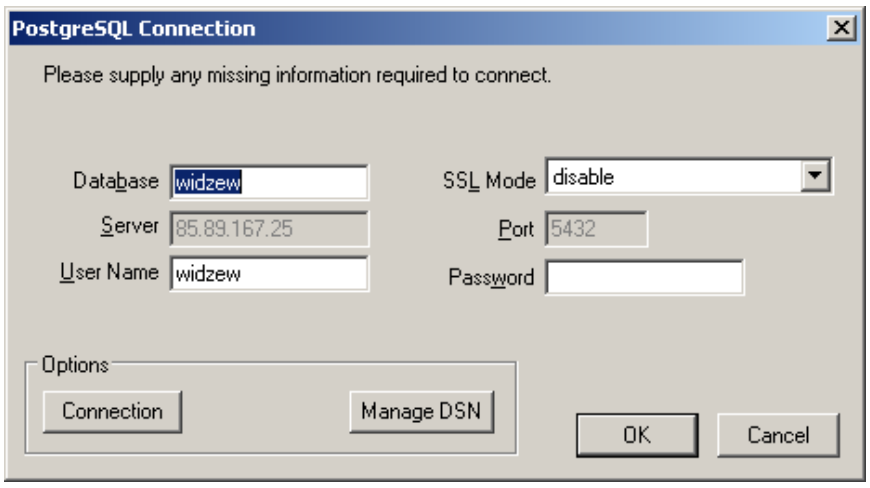
Dwa obszary w centralnej części okna służą odpowiednio do wyboru kwalifikatorów i cech dla których będą generowane podsumowania.



Rysunek 6: Główne okno generatora II (moduł podstawowy)



Rysunek 7: Okno połączenia z bazą, która będzie analizowana

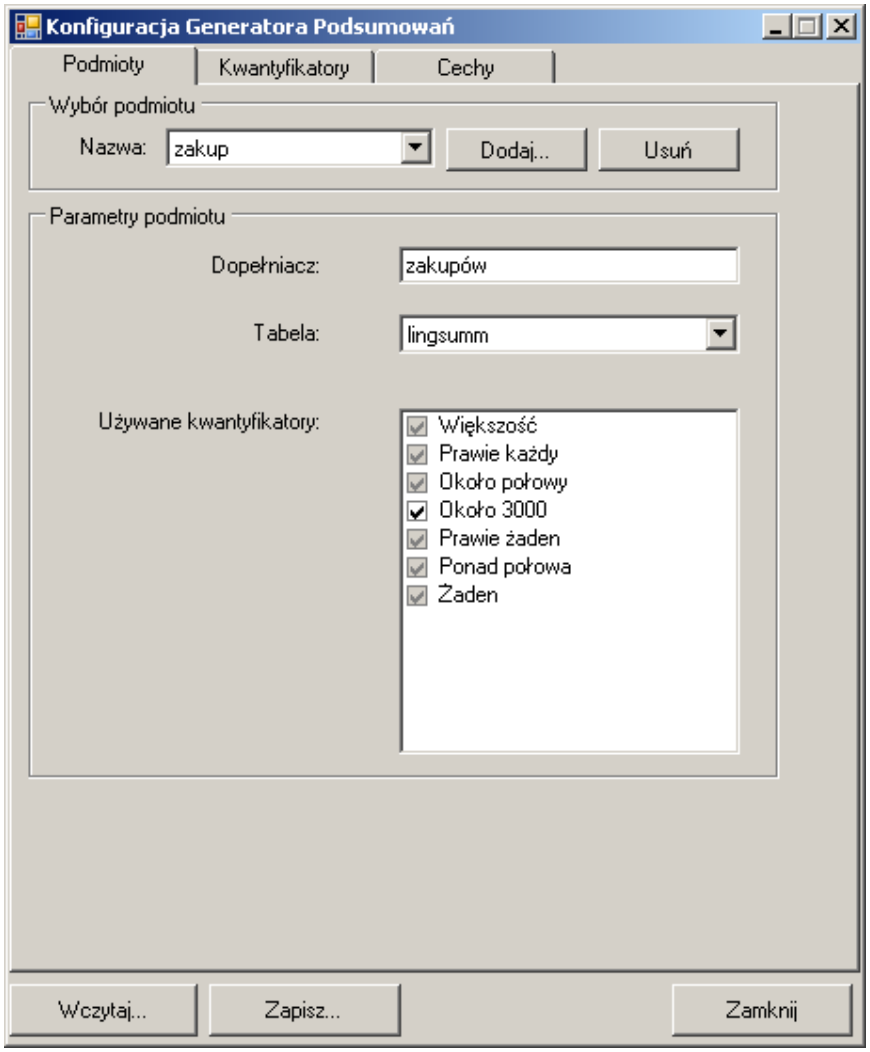


Rysunek 8: Parametry połączenia dla analizowanej bazy PostgreSQL

6 Okno konfiguracji generatora podsumowań

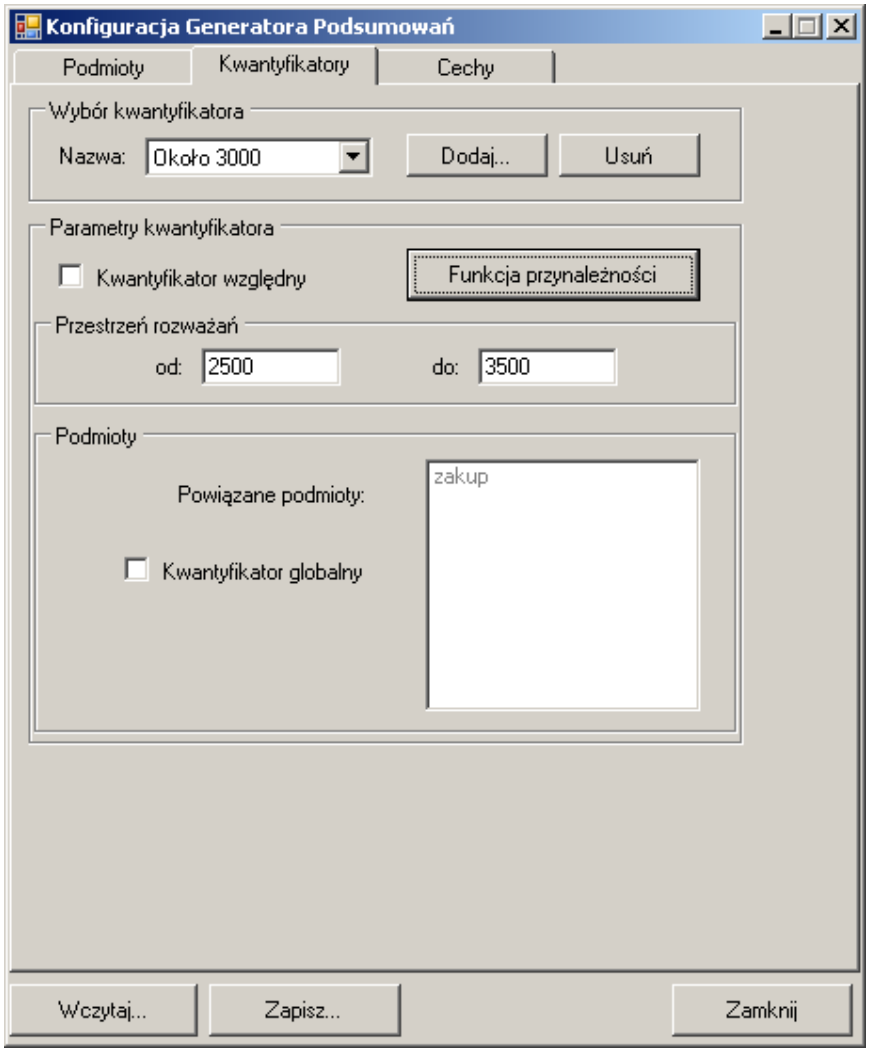
Okno konfiguracji generatora składa się z trzech zakładek: ustawienia podmiotów, kwantyfikatorów oraz cech.

Przedstawia to rysunek 9.

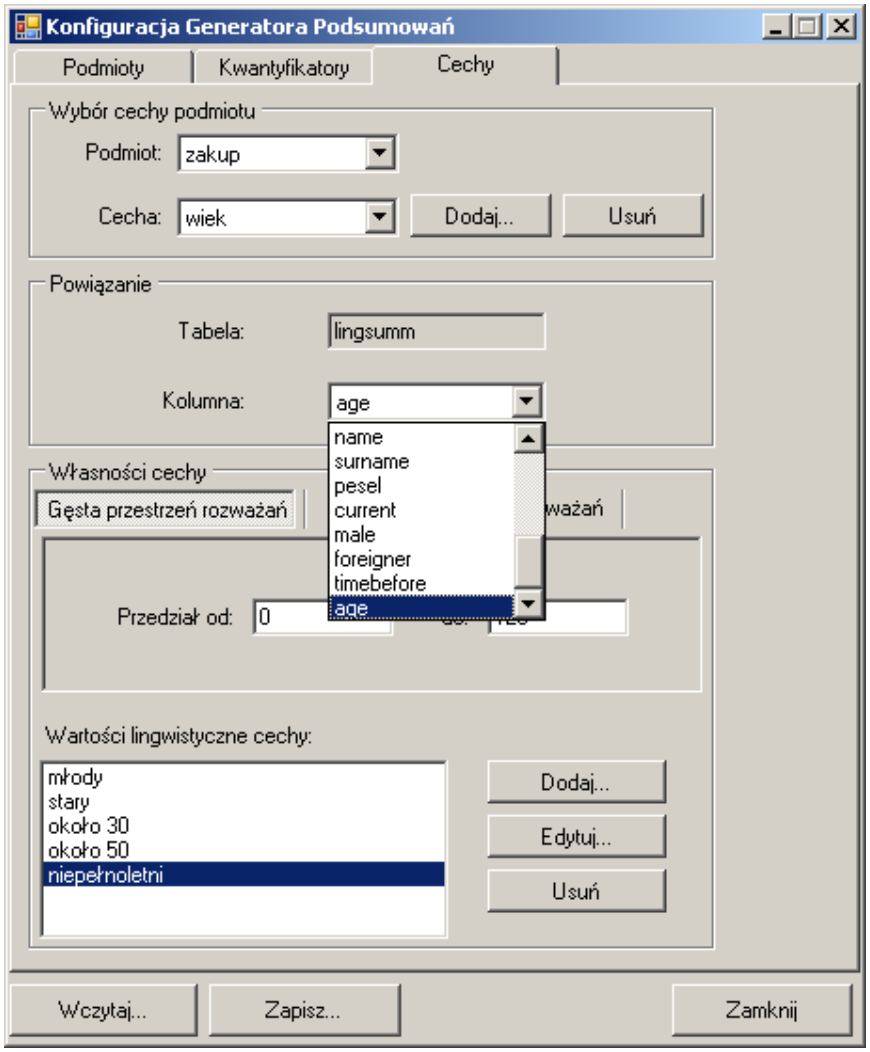


Rysunek 9: Panel konfiguracji podmiotów podsumowań

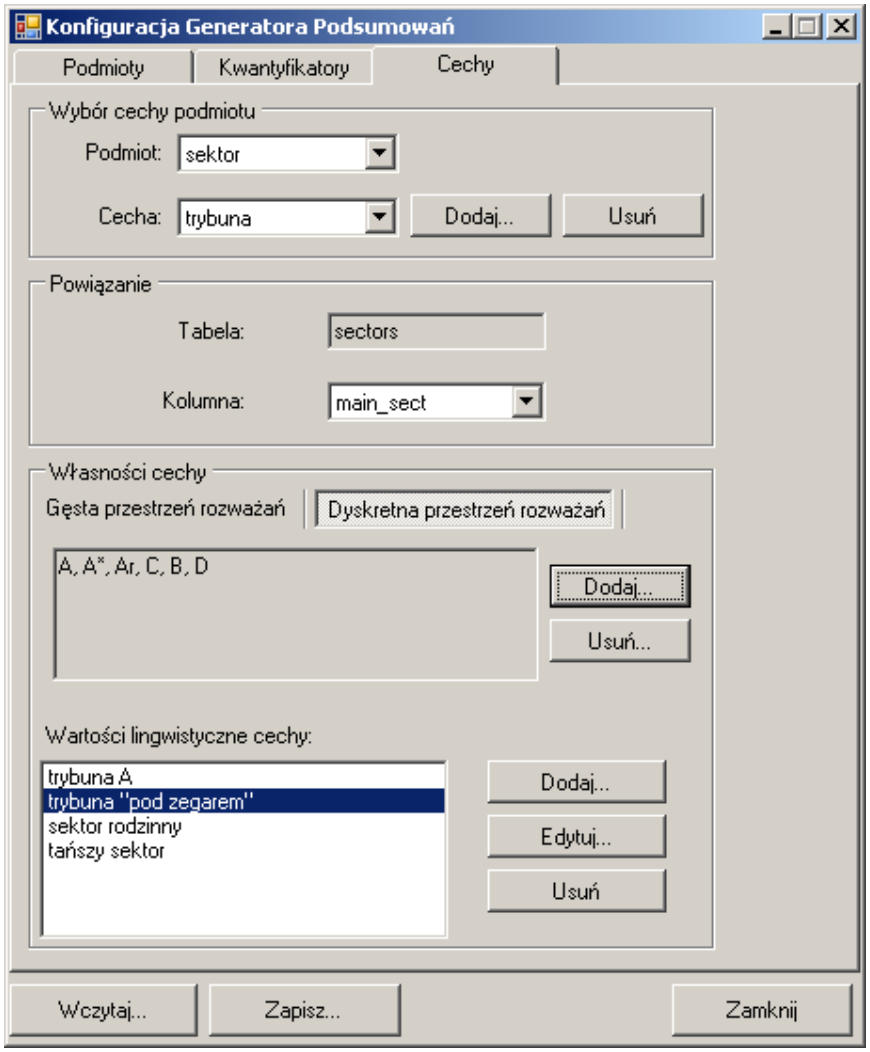
Rysunki 11 i 12 przedstawiają panel konfiguracji cech podmiotów.



Rysunek 10: Panel konfiguracji kwantyfikatorów



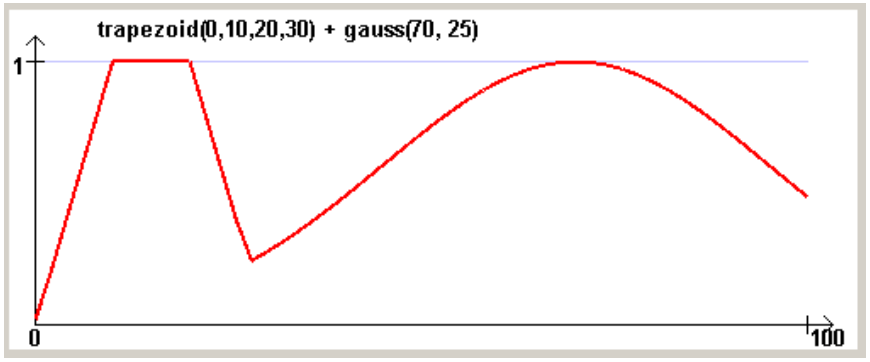
Rysunek 11: Okno konfiguracji cech II



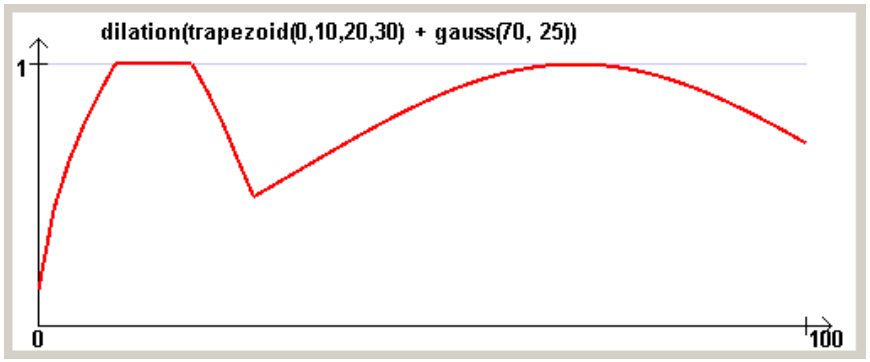
Rysunek 12: Okno konfiguracji cech

7 Wizualizacja funkcji przynależności

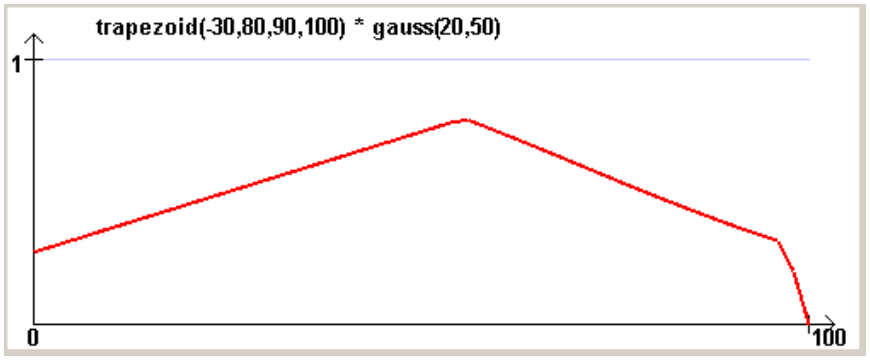
Poniżej przedstawione są wyglądy **stworzonej kontrolki FunctionPlot**, która rysuje wykres funkcji przynależności — dla dowolnej funkcji reprezentowanej przez **abstrakcyjne drzewo składniowe**.



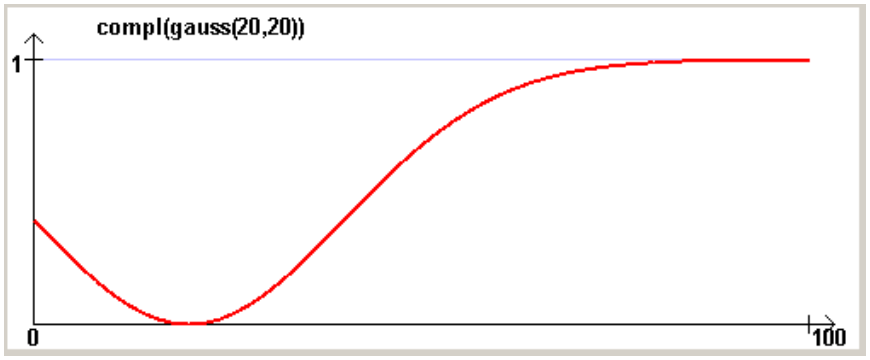
Rysunek 13: Funkcja przynależności trapezoid + gauss



Rysunek 14: Funkcja przynależności dilation(trapezoid + gauss)

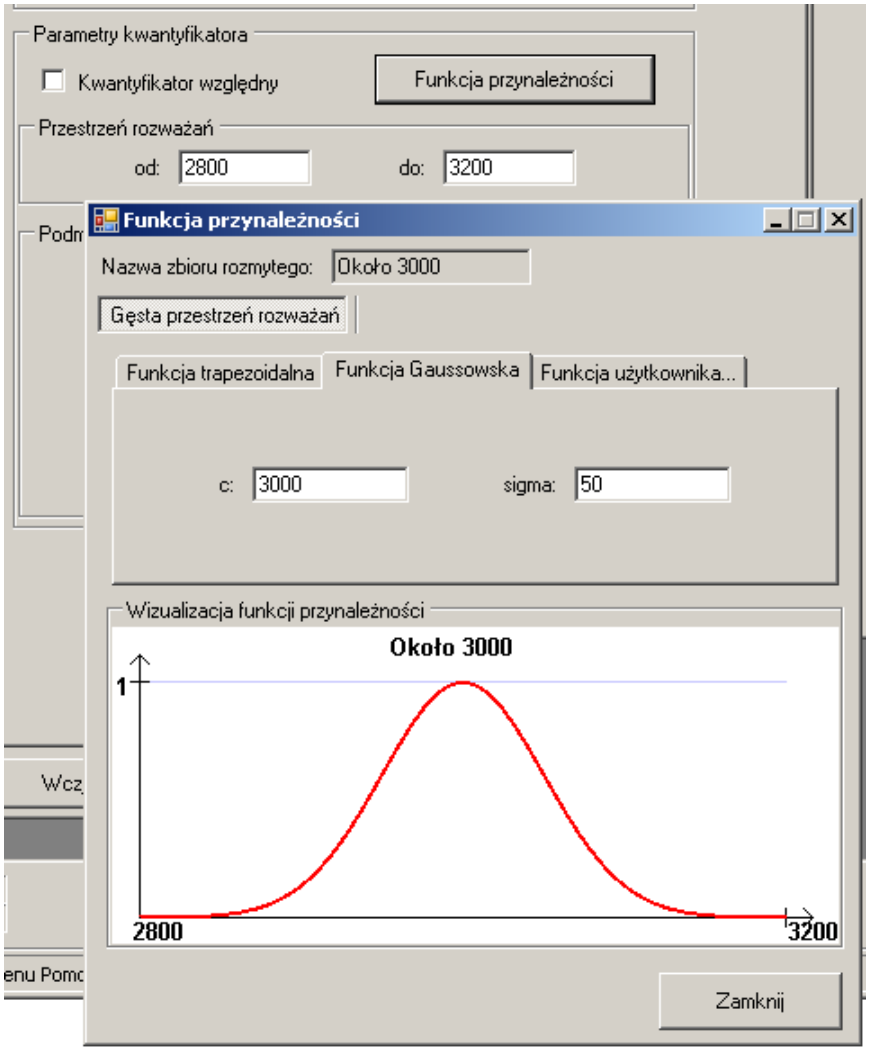


Rysunek 15: Funkcja przynależności trapezoid * gauss

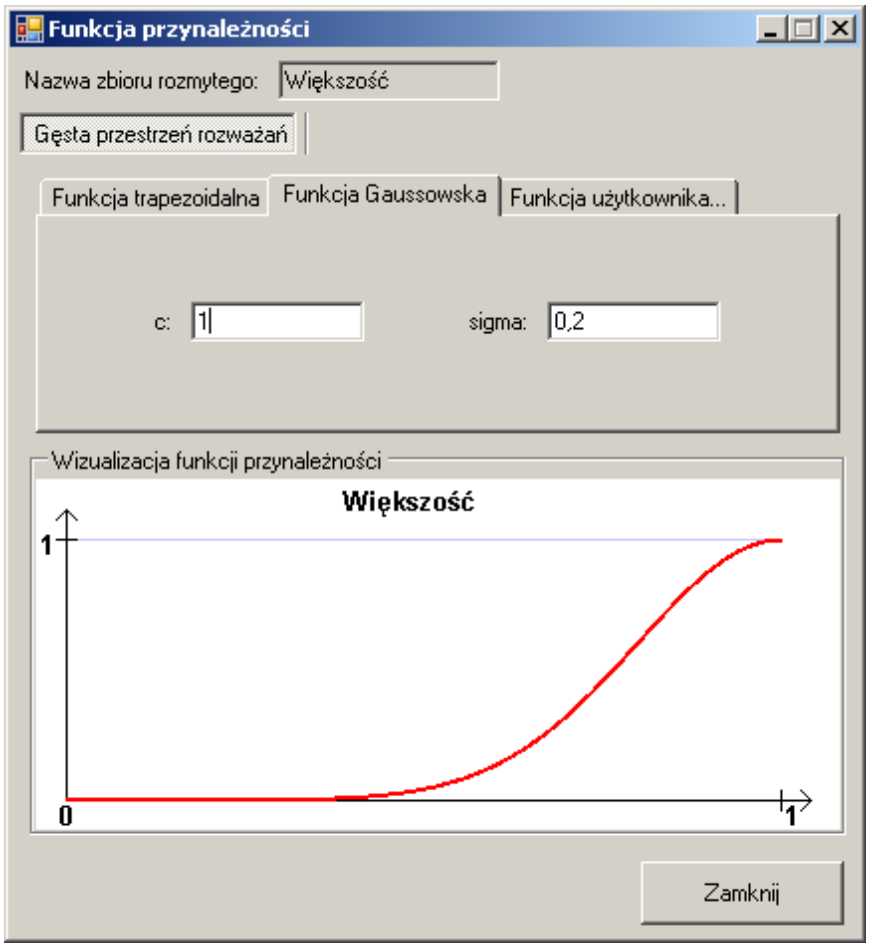


Rysunek 16: Funkcja przynależności complement(gauss)

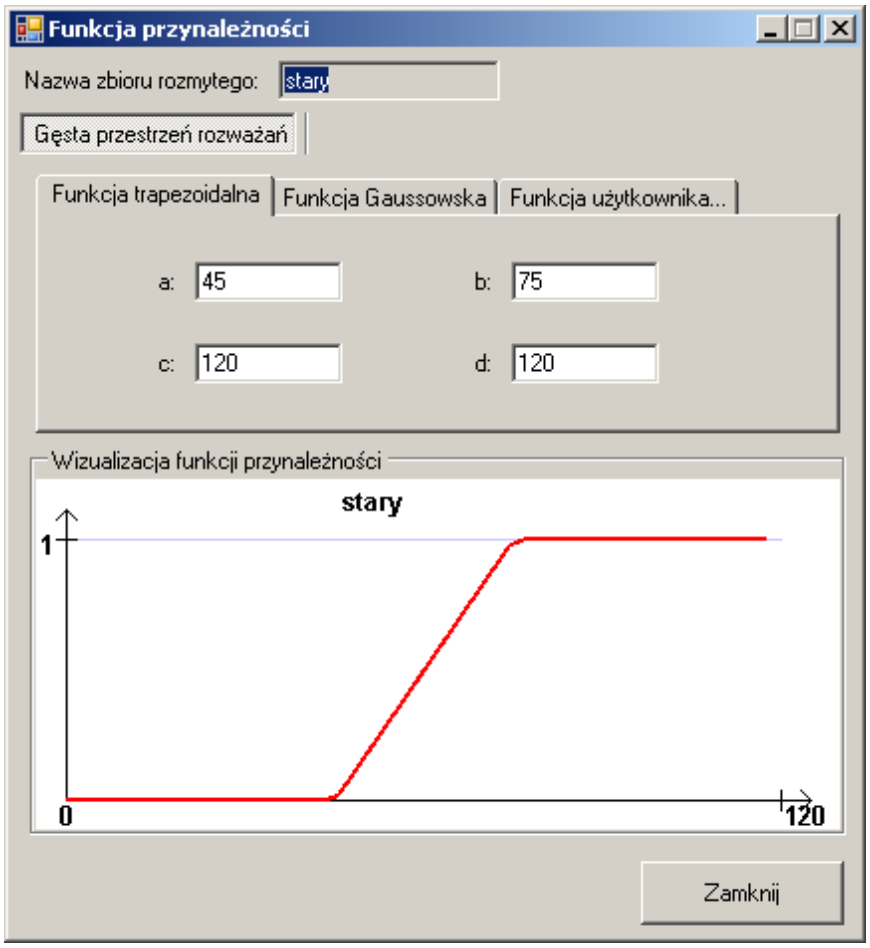
8 Interfejs konfiguratora



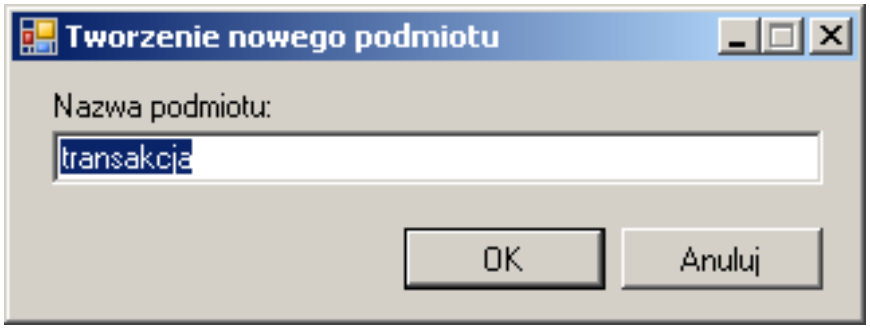
Rysunek 17: Funkcja przynależności kwantyfikatora absolutnego



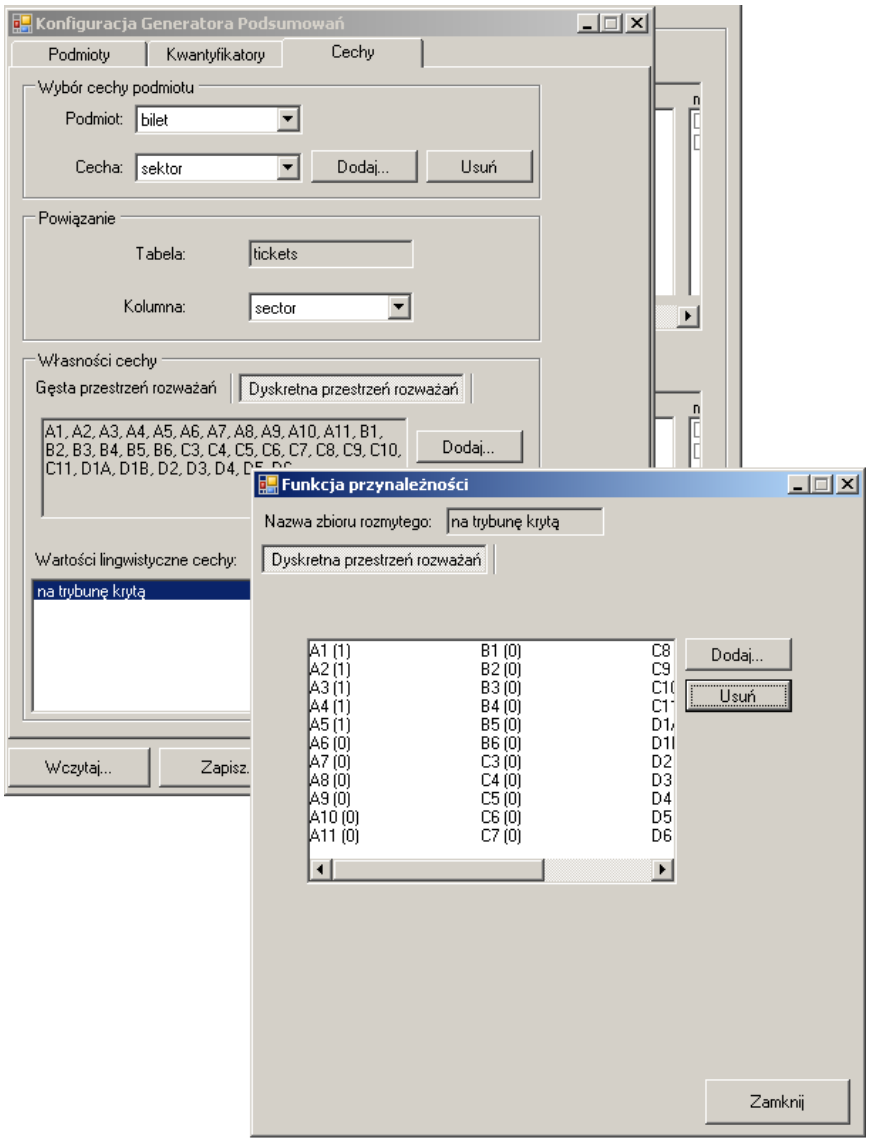
Rysunek 18: Funkcja przynależności kwantyfikatora „większość”



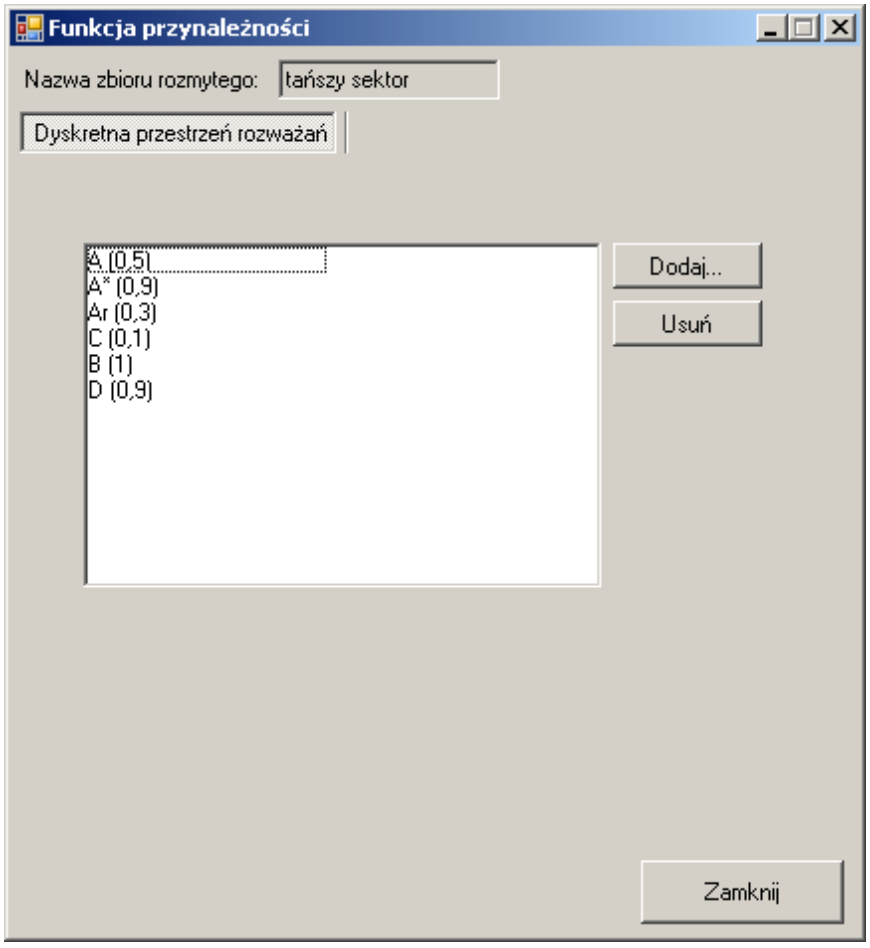
Rysunek 19: Funkcja przynależności wartości lingwistycznej „stary człowiek”



Rysunek 20: Okno dialogowe tworzenia nowego podmiotu



Rysunek 21: Wartość cechy w przestrzeni dyskretnej (sektory i trybuny)

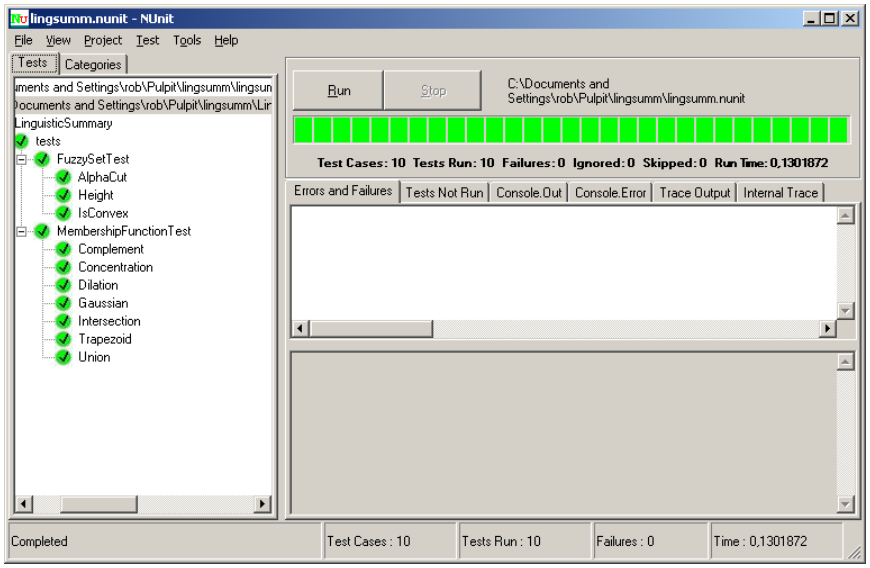


Rysunek 22: Dyskretna funkcja przynależności (trybuny stadionu)

9 Wynik testowania stworzonej biblioteki w NUnit

Dla stworzonych pakietów LinguisticSummary.fuzzy i LinguisticSummary.fuzzy.MembershipFunctions napisaliśmy zestawy testów jednostkowych, do testowania poprawności działania tych pakietów.

Wynik uruchomienia tych testów w NUNit przedstawia rysunek 23.



Rysunek 23: Wynik uruchomienia testów jednostkowych (NUnit)

10 Przykłady wygenerowanych podsumowań

Przykładowa konfiguracja generatora (utworzona całkowicie za pomocą okna konfiguracyjnego) zawiera trzy podmioty: kibice, bilety, zakupy oraz transakcje.

Przykładowe podsumowania wygenerowane dla podmiotu **kibice**:

```
Mało kibiców, jest chuliganem [0,889]
Żaden z kibiców, jest chuliganem [0,095]
Większość kibiców, jest chuliganem [0,103]
Prawie każdy kibiców, jest chuliganem [0,116]
Około połowy kibiców, jest chuliganem [0,110]
Około 3000 kibiców, jest chuliganem [0,101]
Prawie żaden kibiców, jest chuliganem [0,893]
Ponad połowa kibiców, jest chuliganem [0,074]
Dużo kibiców, jest chuliganem [0,090]
Około 1000 kibiców, jest chuliganem [0,111]
```

Podsumowania wygenerowane dla podmiotu **transakcje**:

```
Ponad połowa transakcji Internetowych,
które miała wysoką cenę, odbiór osobisty [0,886]

Mało transakcji Internetowych,
które odbiór osobisty, miała cenę około 100 zł [0,741]

Ponad połowa transakcji Internetowych,
które przesyłka pocztą, miała wysoką cenę [0,880]

Dużo transakcji Internetowych,
które przesyłka pocztą, miała wysoką cenę [0,593]

Większość transakcji Internetowych,
które miała wysoką cenę, odbiór osobisty [0,699]
```

Podsumowania wygenerowane dla podmiotu **bilety**:

```
Żaden biletów, które na sektor rodzinny, jest karnetem [0,928]
Mało biletów, które na sektor rodzinny, jest karnetem [0,922]
```

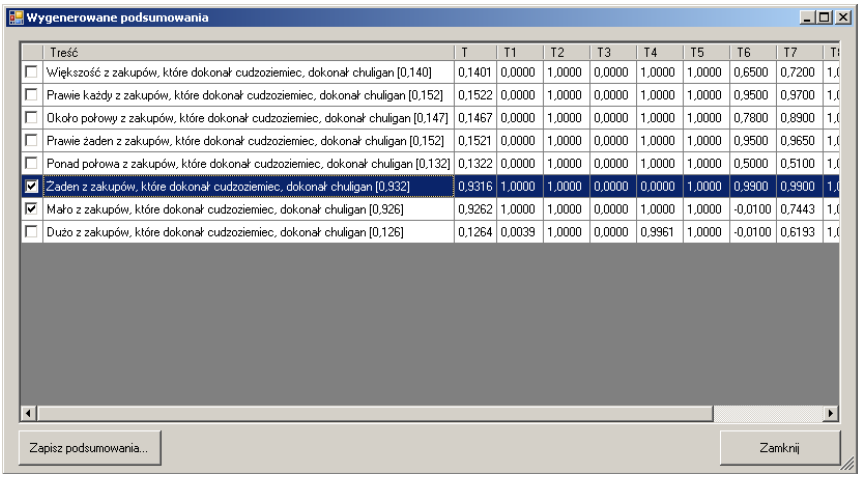
Podsumowania wygenerowane dla podmiotu **zakupy**:

```
Prawie żaden z zakupów,
które dokonał niepełnoletni, dokonał chuligan [0,946]
```

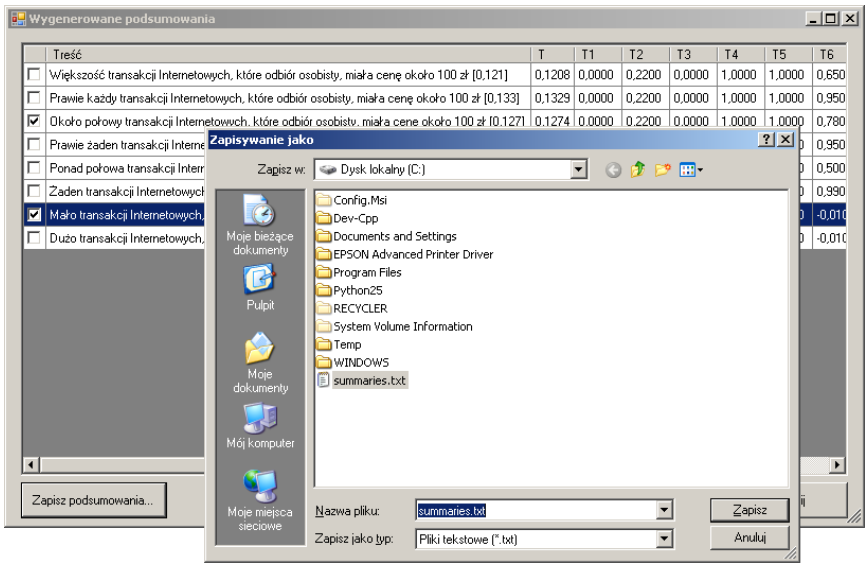
```
Mało z zakupów,
które dokonał niepełnoletni, dokonał chuligan [0,942]

Mało z zakupów,
które dokonał młody człowiek, jest karnetem [0,458]
```

Dodatkowo można byłoby wprowadzić moduł, który przetwarzałby — za pomocą np. wyrażeń regularnych (*regexp*) — wyprodukowane podsumowania, poprawiając je gramatycznie, usuwając powtórzenia itp.

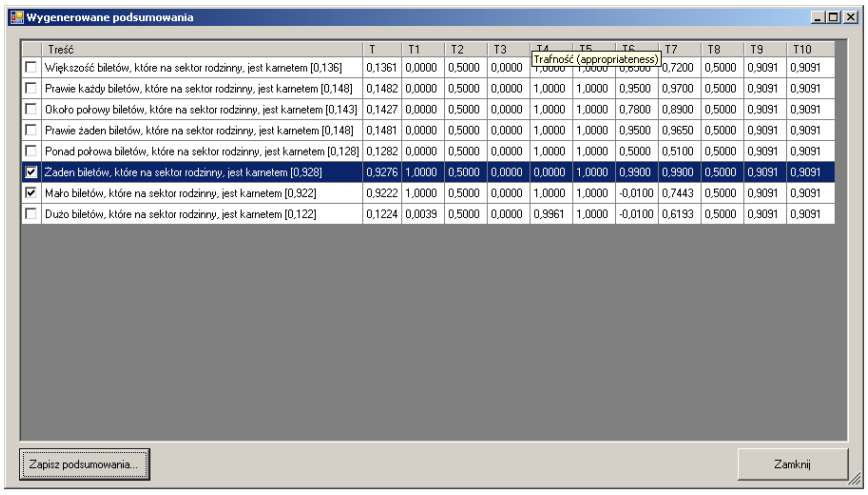
Okna dialogowe z rezultatami wygenerowanych podsumowań dla powyższych konfiguracji przedstawiają rysunki **??**-26.



Rysunek 24: Okno dialogowe wygenerowanych podsumowań i miar



Rysunek 25: Zapis wybranych podsumowań



Rysunek 26: Okno dialogowe wygenerowanych podsumowań i miar