STUDIA INFORMATICA 2008
Volume 29 Number 3A (78)

Przemysław KUDŁACIK
Politechnika Śląska, Instytut Informatyki

OPERACJE NA ZBIORACH ROZMYTYCH Z ODCINKOWO-LINIOWĄ FUNKCJĄ PRZYNALEŻNOŚCI

Streszczenie. Artykuł zawiera szczegółowy opis algorytmów realizujących podstawowe operacje na zbiorach rozmytych z odcinkowo-liniową funkcją przynależności. Funkcja przynależności zbioru reprezentowana jest za pomocą wektora zawierającego kolejne punkty jej opisu. Przedstawione operacje obejmują wyznaczanie wartości przynależności danego elementu zbioru, rozmywanie, wyostrzanie oraz operacje na większej ilości zbiorów, jak suma, przecięcie i agregacja.

Słowa kluczowe: zbiory rozmyte, funkcja odcinkowo-liniowa, algorytmy

OPERATIONS ON FUZZY SETS WITH PIECEWISE LINEAR MEMBERSHIP FUNCTION

Summary. Paper presents precise description of algorithms for basic operations on fuzzy sets with piecewise linear membership function. Membership function is represented with vector storing subsequent points of function's description. Presented algorithms concern operations like calculating a membership value for given element, fuzzyfication, defuzzyfication as well as operations on more than one fuzzy sets like intersection, union and aggregation.

Keywords: fuzzy sets, piecewise linear, algorithms

1. Wprowadzenie

Systemy sztucznej inteligencji oparte na teorii zbiorów rozmytych znajdują coraz szersze zastosowania w wielu obszarach nauki. Sama teoria zbiorów rozmytych oraz zagadnienia związane z opartym na niej wnioskowaniem przybliżonym są silnie rozwijane, o czym świadczy stale rosnąca liczba artykułów z tej dziedziny.

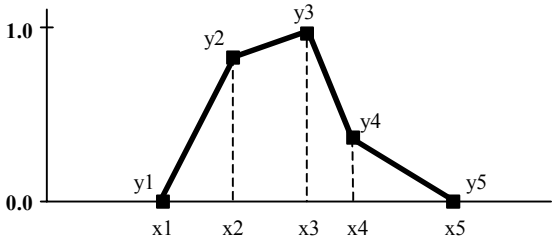
Praktyczne zastosowanie opracowanych teoretycznie rozwiązań wymaga najczęściej samodzielnej implementacji. Natomiast wiele z dostarczanych rozwiązań jest udostępnianych w formacie silnie związanym ze środowiskiem modelowania, którego użyli autorzy. W tym wypadku zastosowanie różnych metod staje się utrudnione, szczególnie dla użytkowników nieposiadających dostępu do wybranego środowiska modelowania lub z niewielkim doświadczeniem w programowaniu. Z tego powodu autor postanowił stworzyć od podstaw bibliotekę, pozwalającą na budowanie i rozwijanie systemów rozmytych w celach naukowych oraz praktycznych zastosowań. Docelowo biblioteka będzie implementowana w popularnych obiektowych językach programowania, jak C++, Java oraz C#. Pierwsze wersje algorytmów zostały zaimplementowane w języku C++. Tym sposobem szeroka liczba użytkowników otrzyma narzędzie tworzenia i rozwijania systemów rozmytych w celach naukowych oraz dla zastosowań praktycznych.

Ze względu na dużą elastyczność oraz prostotę definicji biblioteka wykorzystuje odcinkowo-liniową reprezentację funkcji przynależności zbiorów. Pozwala to na zdefiniowanie dowolnego kształtu funkcji z określoną przez użytkownika dokładnością.

W artykule przedstawiono metody oraz algorytmy podstawowych operacji na zbiorach rozmytych z odcinkowo-liniowymi funkcjami przynależności. Z ich pomocą można budować nawet bardzo skomplikowane systemy, zachowując prostotę opisu oraz możliwość łatwego rozszerzenia o nowe funkcjonalności.

2. Reprezentacja zbioru rozmytego

Rysunek 1 przedstawia przykład liniowo-odcinkowej funkcji przynależności zbioru rozmytego opisanej za pomocą pięciu punktów.



Rys. 1. Zbiór rozmyty z odcinkowo-liniową funkcją przynależności
Fig. 1. Fuzzy set with piecewise linear membership function

Funkcję przynależności można zatem zapisać ogólnie jako

$$\mu(x) = \begin{Bmatrix} y_1 & gdy & x \le x_1 \\ \mu_1(x) & gdy & x_1 < x \le x_2 \\ & ... & \\ \mu_{n-1}(x) & gdy & x_{n-1} < x \le x_n \\ y_n & gdy & x_n < x \end{Bmatrix} \quad (1)$$

gdzie $\mu_i(x)$ dla i=1..n−1 oznacza funkcję liniową opisaną dwoma kolejnymi punktami (x_i,y_i) oraz (x_{i+1},y_{i+1}).

Zatem definicja funkcji przynależności to zbiór uporządkowanych par (x,y), odpowiadających kolejnym punktom opisu.

W omawianej bibliotece kolejne punkty określające funkcję przynależności są przechowywane w jednowymiarowej tablicy prostych struktur definiujących punkt, zawierających składową x oraz y. Składowe punktu są liczbami rzeczywistymi zapisanymi za pomocą wartości zmiennoprzecinkowych podwójnej precyzji[1]. Tabelaryczna forma zapisu danych pozwala na szybkie odwołanie się do wybranej komórki, co zmniejsza czas pracy algorytmów wykonujących operacje na zbiorach. Z drugiej strony, utrudnia ona modyfikacje opisu ze względu na jeden ciągły blok pamięci. Chcąc przykładowo dodać punkt do środka istniejącego opisu, konieczne jest kopiowanie części zawartości tabeli, co znacznie wydłuża czas pracy. Rozwiązaniem mogłoby być zastosowanie innej struktury danych, jak na przykład lista dynamiczna. Charakteryzuje się ona małą złożonością obliczeniową podczas modyfikacji, lecz odwołanie do dowolnego elementu jest dużo bardziej złożone. Dlatego rozwiązanie oparte na tablicy jest bardziej korzystne z punktu widzenia budowanych systemów rozmytych, ponieważ pozwala ono uzyskać mniejszą złożoność obliczeniową podczas pracy. Natomiast opis zbioru jest tworzony zazwyczaj jednorazowo, podczas konfiguracji systemu, więc złożoność obliczeniowa tego procesu ma w tym wypadku znikomy wpływ na czas pracy całości.

Prawie wszystkie algorytmy przedstawione w kolejnych rozdziałach artykułu są napisane w języku C++. Odwołania do punktów opisu zbioru będą wyglądały następująco:

```
punkty[i].x = -1.5;        //przypisanie wartości składowej x punktu i
punkty[i].y = 0.0;         //przypisanie wartości składowej y punktu i
punkty[i] = punkty[i+1];   //przypisanie składowych punktu i+1 opisu do punktu i
```

Zmienna tablicowa o nazwie `punkty` oznacza tabelę jednowymiarową zawierającą opis funkcji przynależności.

[1] Typ **double** w języku C++

3. Wyznaczanie wartości funkcji przynależności

Mając na uwadze tabelaryczną reprezentację opisu przedstawioną w poprzednim rozdziale, wyznaczenie wartości funkcji przynależności w danym punkcie sprowadza się do wyznaczenia wartości funkcji liniowej w odpowiednim przedziale. Algorytm znajdujący właściwy przedział opisany dwoma punktami jest kluczowy, ponieważ od niego zależy złożoność obliczeniowa całego procesu. W omawianej bibliotece do wyszukania właściwego odcinka zastosowano metodę podziału zbioru punktów opisu na dwie części w każdym kolejnym kroku wyszukiwania, co pozwala uzyskać dobrą złożoność logarytmiczną algorytmu.

Wartość y funkcji liniowej w punkcie x , którą opisują dwa punkty (x_1,y_1) oraz (x_2,y_2) wyznaczono według wzoru:

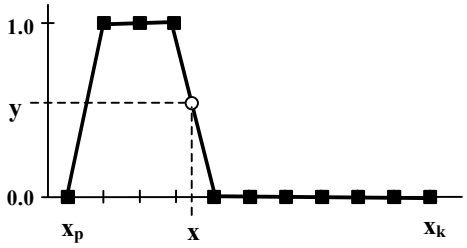
$$y = \frac{(x - x_1)(y_2 - y_1)}{(x_2 - x_1)} + y_1 \tag{2}$$

Poniżej zamieszczono kod tak scharakteryzowanego algorytmu napisany w języku C++:

```
//zmienna 'x' reprezentuje element dla którego obliczana jest wartość funkcji
//zmienna 'rozmiar' zawiera ilość punktów opisujących zbiór

  //zmienne tymczasowe
int lewy;      //pozycja lewego punktu opisu przedziału
int prawy;     //pozycja prawego punktu opisu przedziału
int szerokosc; //ilość punktów pomiędzy lewym i prawym indeksem
int akt;       //aktualny indeks
double wynik;  //wynik do zwrócenia

  //jeśli x znajduje się przed pierwszym punktem opisu to zwróć pierwszą wartość
if ( x <= punkty[0].x ) return punkty[0].y;
  //jeśli x znajduje się za ostatnim punktem opisu to zwróć ostatnią wartość
if ( x >= punkty[rozmiar-1].x ) return punkty[size-1].y;

  //inicjalizacja obejmująca wszystkie indeksy tablicy
lewy  = 0;
prawy = rozmiar-1;

  //dopóki indeksy nie wskazują jednego przedziału wykonuj:
while ( (szerokosc = pra wy - lewy) > 1){
    akt = lewy + szerokosc / 2;         //wyznacz środkowy indeks
    if (x < punkty[akt].x) prawy = akt; //wybierz lewą połowę
                      else lewy  = akt; //wybierz prawą połowę
}
  //wyznacz wartość funkcji liniowej według wzoru 2
wynik = (x - punkty[lewy].x) * (punkty[prawy].y - punkty[lewy].y);
wynik = wynik/(punkty[prawy].x - punkty[lewy].x) + punkty[lewy].y;
return wynik;
```

W bibliotece został również zaimplementowany algorytm dedykowany opisowi funkcji przynależności z równomiernym podziałem dziedziny lub bliskim równomiernemu. W tym wypadku można dokładnie lub z dużą dokładnością wyznaczyć odpowiedni przedział liniowy dzieląc ilość wszystkich zdefiniowanych przedziałów przez stosunek rozmiaru całej dziedzi-

ny do części dziedziny wskazanej danym elementem. Sytuację przedstawia rys. 2 dla zbioru opisanego jedenastoma punktami.



Rys. 2. Opis funkcji przynależności z równomiernym podziałem dziedziny
Fig. 2. Description of a membership function with constant division of the domain

Uwzględniając oznaczenia jak na rys. 2, można zapisać równanie wyznaczające przedział i_P, gdzie *Round* to funkcja zaokrąglająca wartość rzeczywistą do wartości całkowitej.

$$i_P = Round\left(n\frac{x - x_p}{x_k - x_p} \right); \quad x \in (x_p, x_k) \tag{3}$$

Gdy podział dziedziny nieznacznie odbiega od równomiernego, wzór (3) wyznaczy przedział w pobliżu przedziału docelowego. Dlatego w takim wypadku algorytm iteracyjnie poszukuje właściwego przedziału w odpowiednim kierunku począwszy od miejsca wskazanego wzorem. Poniżej pełen algorytm zapisany w języku C++:

```
//zmienna 'x' reprezentuje element dla którego obliczana jest wartość funkcji
//zmienna 'rozmiar' zawiera ilość punktów opisujących zbiór

  //zmienne tymczasowe
int akt;      //aktualna pozycja punktu opisu
int ost;      //ostatnia pozycja punktu opisu
double wynik; //wynik do zwrócenia

  //jeśli x znajduje się przed pierwszym punktem opisu to zwróć pierwszą wartość
if ( x <= punkty[0].x ) return punkty[0].y;
  //jeśli x znajduje się za ostatnim punktem opisu to zwróć ostatnią wartość
if ( x >= punkty[rozmiar-1].x ) return punkty[size-1].y;

  //wyznacz przedział według wzoru 3
akt = (rozmiar-1)*(x - punkty[0].x)/(punkty[rozmiar-1].x - punkty[0].x);

if ( x > punkty[akt].x ){ //poszukuj przedziału na prawo od aktualnego punktu
    ost = akt;
    akt++;
    while ( x > punkty[akt].x ){
        ost = akt;
        akt++;
    }
      //wyznacz wartość funkcji liniowej według wzoru 2
    wynik = (x - punkty[ost].x) * (punkty[akt].y - punkty[ost].y);
    wynik = wynik/(punkty[akt].x - punkty[ost].x) + punkty[ost].y;
    return wynik;
} else {                  //poszukuj przedziału na lewo od aktualnego punktu
    ost = akt;
    act--;
    while ( x < punkty[akt].x ){
        ost = akt;
        act--;
```

```
    }
      //wyznacz wartość funkcji liniowej według wzoru 2;
    wynik = (x - pts[akt].x) * (punkty[ost].y - punkty[akt].y)
    wynik = wynik/(punkty[ost].x - punkty[act].x) + punkty[akt].y;
    return wynik;
    }
}
```

W ogólnym rozwiązaniu algorytm ten będzie mniej efektywny, osiągając w pesymistycznym przypadku złożoność liniową ze względu na pętlę iteracyjnie poszukującą właściwego przedziału.

4. Rozmywanie wartości numerycznej

Najmniej skomplikowaną operacją z algorytmicznego punktu widzenia jest operacja rozmywania, czyli przekształcenia pewnej wartości numerycznej x_n na zbiór rozmyty [2-8]. W tym celu należy zdefiniować zbiór rozmyty w ten sposób, aby jego funkcja przynależności była opisana wokół środka x_s=0. Zbiór ten w omawianej bibliotece nosi nazwę zbioru rozmywającego.

Chcąc rozmyć numeryczną wartość wejściową x_n, należy utworzyć nowy zbiór rozmyty na podstawie zbioru rozmywającego, dodając do wszystkich składowych x punktów opisu funkcji przynależności wartość rozmywaną x_n. Utworzona w ten sposób funkcja przynależności nowego zbioru odpowiada funkcji zbioru rozmywającego, lecz jest ona określona wokół wejściowej wartości numerycznej x_n.

```
  //przesuń składowe x punktów opisu zbioru o wartość rozmywaną 'xn'
for (i=0; i<ilosc_punktow; i++){
    punkty[i].x += xn;
}
```

Algorytm realizujący to zadanie ma złożoność liniową zależną od rozmiaru zbioru rozmywającego, ponieważ musi przetworzyć wszystkie punkty opisu funkcji przynależności.

5. Wyostrzanie zbioru rozmytego

W niniejszym punkcie zostanie przedstawionych kilka algorytmów odpowiadających najpopularniejszym metodom wyostrzania, które zaimplementowano w omawianej bibliotece. Ich wspólną cechą jest liniowa złożoność obliczeniowa zależna od ilości punktów definiujących funkcję przynależności, ponieważ w celu obliczenia wyniku konieczne jest przeanalizowanie wszystkich punktów opisu.

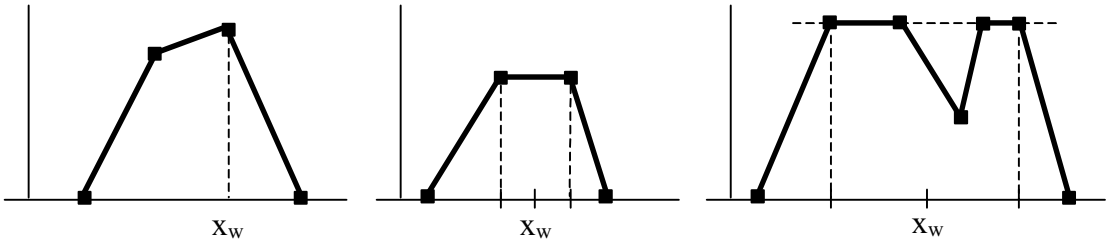
5.1. Wyostrzanie zbioru rozmytego metodą maksimum

Najprostszą metodą wyostrzania zbioru rozmytego jest metoda zwracająca element, dla którego funkcja przynależności osiąga maksimum [2-8]. Wartość wyostrzana będzie oznaczana jako x_w.

W przypadku gdy wartość maksymalna funkcji odpowiada większej liczbie elementów dziedziny, wybierana jest wartość środkowa pomiędzy dwoma skrajnymi elementami tej grupy.

$$x_w = \frac{\inf(X_{MAX}) + \sup(X_{MAX})}{2} \tag{4}$$

gdzie X_{MAX} oznacza zbiór wszystkich elementów dziedziny, dla których funkcja przynależności $\mu(x)$ osiąga wartość maksymalną.



Rys. 3. Wyostrzanie zbioru rozmytego metodą maksimum
Fig. 3. Defuzzyfication of a fuzzy set with maximum method

Algorytm realizujący tę operację w omawianej bibliotece ma następującą postać:

```
  //zmienne tymczasowe
int i, iLewy, iPrawy;  //licznik oraz indeksy skrajnych elementów maksymalnych

iLewy = iPrawy = 0;
for (i=1; i<rozmiar; i++){ //pętla dla wszystkich punktów
   if (punkty[iLewy].y < punkty[i].y) { //jeśli punkt jest maksimum
      iLewy = iPrawy = i;
   } else                                //jeśli punkt równy aktualnemu maksimum
       if (punkty[iLewy].y == punkty[i].y) iPrawy = i;
}
  //wynik zgodnie z wzorem 4
return (punkty[iLewy].x + punkty[iPrawy].x) / 2.0;
```

5.2. Wyostrzanie zbioru rozmytego metodą COG

Najpopularniejszą metodą wyostrzania jest metoda środka grawitacji oznaczana skrótem COG[2] [2-8]. W omawianej bibliotece jest ona domyślną metodą wyostrzania zbiorów rozmytych.

Zgodnie z definicją wartość wyostrzoną zbioru opisanego funkcją przynależności $\mu(x)$ wyznacza się w następujący sposób:

[2] Ang. Center Of Gravity

$$x_w = \frac{\int_X x\mu(x)dx}{\int_X \mu(x)dx}, \qquad x \in X \tag{5}$$

W przypadku liniowej aproksymacji wartości całek powyższego wyrażenia można wyznaczyć iteracyjnie, przetwarzając kolejne liniowe segmenty opisane w przestrzeniach X_i funkcjami $\mu_i(x)$, dla i=1..n.

$$\int_X x\mu(x)dx = \sum_{i=1}^{n} \int_{X_i} x\mu_i(x)dx \tag{6}$$

$$\int_X \mu(x)dx = \sum_{i=1}^{n} \int_{X_i} \mu_i(x)dx \tag{7}$$

Każdy segment funkcji $\mu(x)$ może zostać opisany równaniem prostej przechodzącej przez dwa punkty (x_1,y_1) oraz (x_2,y_2) :

$$(y - y_1)(x_2 - x_1) = (x - x_1)(y_2 - y_1) \tag{8}$$

W celu obliczenia całki bardziej przydatna jest postać kierunkowa równania prostej:

$$y = ax + b \tag{9}$$

Parametry a i b wyznaczone z równania (8) mają następującą postać:

$$a = \frac{(y_2 - y_1)}{(x_2 - x_1)} \tag{10}$$

$$b = \frac{(x_2 y_1 - x_1 y_2)}{(x_2 - x_1)} \tag{11}$$

Podstawiając równanie (9) do wzoru (6) dla jednego segmentu opisanego punktami (x_1,y_1), (x_2,y_2) otrzymuje się :

$$\int_{x_1}^{x_2} x(ax + b)dx = \int_{x_1}^{x_2} (ax^2 + bx)dx = \frac{ax_2^3}{3} + \frac{bx_2^2}{2} - \frac{ax_1^3}{3} - \frac{bx_1^2}{2} =$$

$$= \frac{a}{3}\left(x_2^3 - x_1^3\right) + \frac{b}{2}\left(x_2^2 - x_1^2\right) \tag{12}$$

Analogicznie, wyznaczając całkę z równania (7) dla jednego segmentu, otrzymuje się:

$$\int_{x_1}^{x_2} (ax + b)dx = \frac{a}{2}\left(x_2^2 - x_1^2\right) + b\left(x_2 - x_1\right) \tag{13}$$

Podstawiając wyrażenia (12) i (13) do równań (6) i (7), otrzymuje się:

$$\int_X x\mu(x)dx = \sum_{i=1}^{n}\left(\frac{a}{3}\left(xi_2^3 - xi_1^3\right) + \frac{b}{2}\left(xi_2^2 - xi_1^2\right)\right) \tag{14}$$

$$\int_X \mu(x)dx = \sum_{i=1}^{n}\left(\frac{a}{2}\left(xi_2^2 - xi_1^2\right) + b\left(xi_2 - xi_1\right)\right) \tag{15}$$

Ostateczną postać równania opisującego wyostrzanie otrzymuje się po podstawieniu powyższych dwóch równań do wyrażenia (5)

$$x_w = \frac{\sum_{i=1}^{n}\left(\frac{a}{3}\left(xi_2^3 - xi_1^3\right) + \frac{b}{2}\left(xi_2^2 - xi_1^2\right)\right)}{\sum_{i=1}^{n}\left(\frac{a}{2}\left(xi_2^2 - xi_1^2\right) + b\left(xi_2 - xi_1\right)\right)} \tag{16}$$

Rozwijając parametry a i b zgodnie ze wzorami (10) i (11) oraz upraszczając względem stałych i $(x_2\text{-}x_1)$ otrzymuje się:

$$x_w = \frac{\sum_{i=1}^{n}\left(\frac{2}{3}\left(yi_2 - yi_1\right)\left(xi_2^3 - xi_1^3\right) + \left(xi_2 yi_1 - xi_1 yi_2\right)\left(xi_2^2 - xi_1^2\right)\right)}{\sum_{i=1}^{n}\left(\left(yi_2 - yi_1\right)\left(xi_2^2 - xi_1^2\right) + 2\left(xi_2 yi_1 - xi_1 yi_2\right)\left(xi_2 - xi_1\right)\right)} \tag{17}$$

W celu zmniejszenia złożoności obliczeniowej algorytmu te same wyrażenia składowe występujące w równaniu (17) wyznaczane są jednokrotnie. Ponadto operacje mnożenia przez stałą przeniesione są poza sumę. Tym sposobem otrzymuje się bardziej efektywną numerycznie postać

$$x_w = \frac{\frac{2}{3}\sum_{i=1}^{n} dy_i dx_i^3 + \sum_{i=1}^{n} dd_i dx_i^2}{\sum_{i=1}^{n} dy_i dx_i^2 + 2\sum_{i=1}^{n} dd_i dx_i} \tag{18}$$

gdzie $dy_i=yi_2\text{-}yi_1$, $dx_i=xi_2\text{-}xi_1$, $dx_i^2=xi_2^2\text{-}xi_1^2$, $dx_i^3=xi_2^3\text{-}xi_1^3$ oraz $dd_i=xi_2yi_1\text{-}xi_1yi_2$.

Algorytm wyostrzający zbiór zgodnie ze wzorem (18) ma następującą postać:

```
  //zmienne tymczasowe
int i;                                //licznik
double dx,dx2,dx3,dy,dd;              //różnice
double x12,x13,x22,x23;               //potęgi zmiennych x1 i x2
double Sdydx3,Sdddx2,Sdydx2,Sdddx; //sumy

Sdydx3 = Sdddx2 = Sdydx2 = Sdddx = 0.0; //zeruj sumy

  //pętla po wszystkich przedziałach funkcji przynależności
for (i=0; i<(rozmiar-1); i++){
      //oblicz krótkie wyrażenia składowe
    dx  = punkty[i+1].x - punkty[i].x;   // oblicz x2-x1
    dy  = punkty[i+1].y - punkty[i].y;   // oblicz y2-y1
    x12 = punkty[i].x * punkty[i].x;     // oblicz x1^2
    x13 = x12 * punkty[i].x;             // oblicz x1^3
    x22 = punkty[i+1].x * punkty[i+1].x; // oblicz x2^2
    x23 = x22 * punkty[i+1].x;           // oblicz x2^3
    dx2 = x22 - x12;                     // oblicz x2^2 - x1^2
    dx3 = x23 - x13;                     // oblicz x2^3 - x1^3
                                         // oblicz x2*y1 - x1*y2
    dd  = punkty[i+1].x * punkty[i].y - punkty[i].x * punkty[i+1].y;
      //aktualizuj sumy lokalne
```

```
        Sdydx3 += dy * dx3 / dx;
        Sdddx2 += dd * dx2 / dx;
        Sdydx2 += dy * dx2 / dx;
        Sdddx  += dd * dx  / dx;
    }
      //oblicz ostateczny wynik
    return ( Sdydx3/3.0 + Sdddx2/2.0 )/( Sdydx2/2.0 + Sdddx );
```

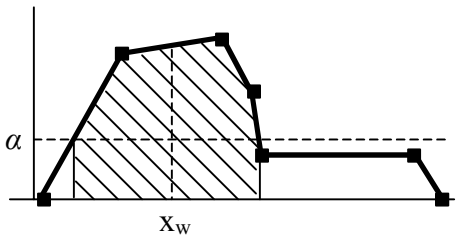
Dla uproszczenia algorytmu zostały z niego usunięte dwie części sprawdzające. Pierwsza pozwala pominąć bardzo krótkie przedziały, które powstają podczas przetwarzania zbioru w wyniku błędów zaokrąglenia granic przedziałów. Taka operacja praktycznie nie wpływa na poprawny rezultat. Chroni go jednak przed zaburzeniem, które powstałoby w wyniku dzielenia przez bardzo małą wartość, będącą na granicy rozdzielczości liczby zmiennoprzecinkowej. Druga część sprawdzająca zapobiega dzieleniu przez zero w przypadku próby wyostrzenia funkcji z zerowym polem pod wykresem.

5.3. Wyostrzanie zbioru rozmytego metodą ICOG

Kolejną zaimplementowaną w bibliotece metodą wyostrzania jest indeksowana metoda środka ciężkości oznaczana skrótem ICOG[3] [5]. Metoda pozwala na wyeliminowanie wartości funkcji przynależności będących poniżej zadanego progu α.

$$x_w = \frac{\int_X x\mu_\alpha(x)dx}{\int_X \mu_\alpha(x)dx}, \qquad x \in X, \quad \mu_\alpha(x) = \begin{cases} \mu(x) & gdy\ \mu(x) \geq \alpha \\ 0 & gdy\ \mu(x) < \alpha \end{cases} \tag{19}$$

Zasadę przedstawia rys. 4, na którym zaznaczono wyjściowe pole uwzględniane podczas wyostrzania.



Rys. 4. Wyostrzanie zbioru rozmytego indeksowaną metodą środka ciężkości
Fig. 4. Fuzzy set defuzzyfication with method of indexed center of gravity

Zatem algorytm wyostrzania opiera się na przedstawionym w punkcie 5.2 algorytmie COG, lecz uwzględnia tylko te punkty opisu funkcji przynależności, których wartość jest większa od progu α. Dodatkowo wyznaczane są punkty przecięcia funkcji przynależności z poziomem α, aby nie pominąć ważnych obszarów granicznych.

[3] Ang. Indexed Center Of Gravity

```
  //zmienne tymczasowe
int i;                              //licznik
double dx,dx2,dx3,dy,dd;            //różnice
double x12,x13,x22,x23;             //potęgi zmiennych x1 i x2
double Sdydx3,Sdddx2,Sdydx2,Sdddx; //sumy
double tmpx1,tmpx2,tmpy1,tmpy2;     //zmienione x1,y1,x2,y2

Sdydx3 = Sdddx2 = Sdydx2 = Sdddx = 0.0;

  //pętla po wszystkich przedziałach funkcji przynależności
for (i=0; i<(rozmiar-1); i++){
      //przygotuj dane do modyfikacji
    tmpx1 = punkty[i].x; tmpx2 = punkty[i+1].x;
    tmpy1 = pts[i].y; tmpy2 = pts[i+1].y;
      //jeśli punkt startowy i końcowy < alpha to omijaj przedział
    if ( tmpy1>=alpha || tmpy2>=alpha ) {
        if ( tmpy1<alpha ) {          //przecięcie, oblicz nowy punkt startowy
          tmpx1 = tmpx1 + (tmpx2-tmpx1)*(alpha-tmpy1)/(tmpy2-tmpy1);
          tmpy1 = alpha;
        }
        if ( tmpy2<alpha ) {          //przecięcie, oblicz nowy punkt końcowy
          tmpx2 = tmpx2 - (tmpx2-tmpx1)*(alpha-tmpy2)/(tmpy1-tmpy2);
          tmpy2 = alpha;
        }
          //oblicz jeden krok metody COG dla przygotowanych parametrów
        dx  = tmpx2 - tmpx1;                 // x2-x1
        dy  = tmpy2 - tmpy1;                 // y2-y1
        x12 = tmpx1 * tmpx1;                 // x1^2
        x13 = x12 * tmpx1;                   // x1^3
        x22 = tmpx2 * tmpx2;                 // x2^2
        x23 = x22 * tmpx2;                   // x2^3
        dx2 = x22 - x12;                     // x2^2 - x1^2
        dx3 = x23 - x13;                     // x2^3 - x1^3
        dd  = tmpx2 * tmpy1 - tmpx1 * tmpy2; // x2*y1 - x1*y2
        Sdydx3 += dy * dx3 / dx;
        Sdddx2 += dd * dx2 / dx;
        Sdydx2 += dy * dx2 / dx;
        Sdddx  += dd * dx  / dx;
    }
}
  //oblicz ostateczny wynik
return ( Sdydx3/3.0 + Sdddx2/2.0 )/( Sdydx2/2.0 + Sdddx );
```

5.4. Wyostrzanie zbioru rozmytego metodą MICOG

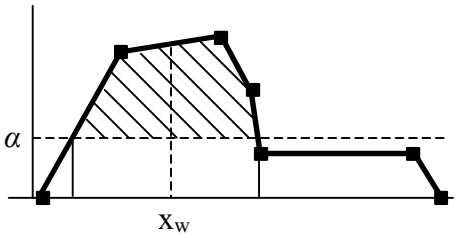
Ostatnią metodą dostępną w bibliotece jest zmodyfikowana indeksowana metoda środka ciężkości MICOG[4] [5].

Podobnie do metody indeksowanej pozwala ona na eliminację wartości funkcji przynależności znajdujących się poniżej zadanego progu α. Różnica polega na odjęciu wartości progu od funkcji przynależności.

$$x_w = \frac{\int_X x\mu_\alpha(x)dx}{\int_X \mu_\alpha(x)dx}, \quad x \in X, \quad \mu_\alpha(x) = \begin{cases} \mu(x)-\alpha & gdy\ \mu(x)-\alpha \geq 0 \\ 0 & gdy\ \mu(x)-\alpha < 0 \end{cases} \tag{20}$$

[4] Ang. Modified Indexed Center Of Gravity

Zasadę przedstawia rys. 5, na którym zaznaczono wyjściowe pole uwzględniane podczas wyostrzania.



Rys. 5. Wyostrzanie zbioru rozmytego modyfikowaną indeksowaną metodą środka ciężkości
Fig. 5. Fuzzy set defuzzyfication with method of modified indexed center of gravity

Algorytm wyostrzania metodą MICOG jest bardzo podobny do algorytmu metody ICOG, ponieważ przetwarzanie wykonywane jest również dla przedziałów funkcji przynależności, w których $\mu(x)>\alpha$. Jedyną różnicą jest odjęcie progu α od każdej składowej y przetwarzanego punktu opisu. Z tego powodu poniżej przedstawiono jedynie zmienioną część kodu algorytmu ICOG.

```
...
  //przygotuj dane do modyfikacji zgodnie z parametrem alpha
tmpx1 = punkty[i].x; tmpx2 = punkty[i+1].x;
tmpy1 = pts[i].y - alpha; tmpy2 = pts[i+1].y - alpha;
  //jeśli punkt startowy i końcowy < alpha to omijaj przedział
if ( tmpy1>=0.0 || tmpy2>=0.0 ) {
    if ( tmpy1<0.0 ) {            //przecięcie, oblicz nowy punkt startowy
        tmpx1 = tmpx1 - (tmpx2-tmpx1)*tmpy1/(tmpy2-tmpy1);
        tmpy1 = 0.0;
    }
    if ( tmpy2<0.0 ) {            //przecięcie, oblicz nowy punkt końcowy
        tmpx2 = tmpx2 + (tmpx2-tmpx1)*tmpy2/(tmpy1-tmpy2);
        tmpy2 = 0.0;
    }
...
```

6. Przecięcie i suma zbiorów rozmytych

Operacje przecięcia i sumy zbiorów rozmytych można wykonać na zbiorach opisanych w tej samej przestrzeni X

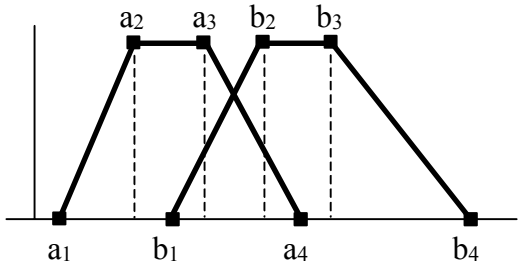
$$A \cap B = C, \quad \mu_C(x) = \mu_A(x) *_T \mu_B(x), \quad \forall_{x \in X} \tag{21}$$

$$A \cup B = C, \quad \mu_C(x) = \mu_A(x) *_S \mu_B(x), \quad \forall_{x \in X} \tag{22}$$

gdzie $*_T$ oraz $*_S$ to odpowiednio dowolna t-norma lub s-norma.

W obu przypadkach poziom przynależności elementu x zbioru wyjściowego jest równy wynikowi operacji na poziomach przynależności tego samego elementu x zbiorów wejściowych. Operację definiuje zatem funkcja dwóch zmiennych, jak np. t-norma minimum lub s-norma maksimum. Omawiana biblioteka zawiera implementację kilku popularnych rodzajów t-norm i s-norm najczęściej spotykanych w literaturze [1-9].

Realizując operację przecięcia lub sumy w przypadku liniowej aproksymacji funkcji przynależności, należy uwzględnić możliwość opisu każdego zbioru dla różnych elementów dziedziny *X*. Sytuację ilustruje rys. 6.

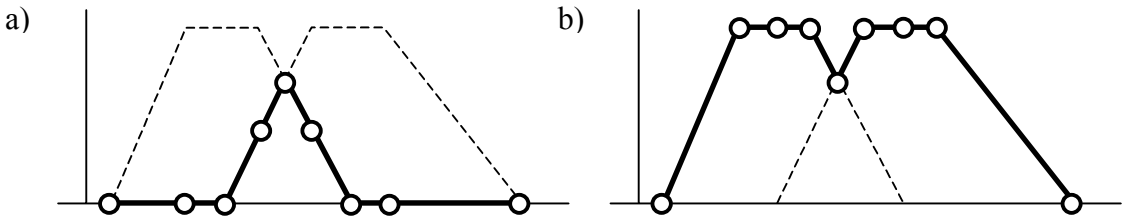


Rys. 6. Funkcje przynależności dwóch zbiorów rozmytych
Fig. 6. Membership functions of two fuzzy sets

Algorytm realizujący tak zdefiniowane zadanie powinien wykonać określoną operację dla wszystkich elementów opisu funkcji przynależności obydwu zbiorów. Wersja zaimplementowana w omawianej bibliotece przetwarza kolejne punkty zbiorów począwszy od punktu o najmniejszej składowej *x* (pierwszy punkt opisu z lewej strony).

Dla wybranego elementu opisanego punktem (x_j,y_j) obliczana jest wartość funkcji przynależności drugiego zbioru dla tego samego elementu x_j. Następnie na obydwu wartościach funkcji przynależności wykonywana jest operacja, której wynik określa nowy poziom przynależności y_n. Tym sposobem otrzymuje się parę x_j,y_n tworzącą kolejny, nowy punkt opisu zbioru wynikowego. Ponadto, algorytm wykonuje operację również w punktach przecięcia funkcji przynależności, dodając tym samym charakterystyczne miejsca opisu.

Stosując algorytm do przypadku z rys. 6 kolejno zostałyby przetworzone elementy dziedziny opisane punktami a_1, a_2, b_1, a_3, punkt przecięcia funkcji, b_2, a_4, b_3 i ostatni b_4. Tym sposobem uzyskałoby się zbiór wynikowy opisany za pomocą dziewięciu punktów.



Rys. 7. Wynik operacji na zbiorach rozmytych: a) przecięcie, b) suma
Fig. 7. Result of an operation on fuzzy sets: a) intersection, b) union

Rysunek 7 przedstawia wyniki przecięcia oraz sumy z użyciem odpowiednio operacji minimum i maksimum dla przykładu z rys. 6. Okręgi oznaczają punkty opisu zbioru wynikowego. Należy zwrócić uwagę na powstanie płaskich odcinków opisanych większą ilością punktów. Problem ten zostanie omówiony w dalszej części artykułu.

Ponieważ pełen algorytm operacji na dwóch zbiorach rozmytych zapisany w języku C++ jest obszerny, w celu zapewnienia odpowiedniej przejrzystości zostanie on przedstawiony w ogólniejszej postaci:

```
1. Pobierz pierwszy z lewej punkt opisu zbiorów A i B
2. Oblicz wartość funkcji przynależności dla drugiego zbioru
3. Wykonaj operację na poziomach przynależności i dodaj punkt do opisu wyniku
4. Dopóki istnieją nie przetworzone punkty opisu zbiorów A lub B wykonuj :
    4.1. Pobierz kolejny punkt opisu
    4.2. Oblicz wartość funkcji przynależności dla drugiego zbioru
    4.3. Jeśli nastąpiło przecięcie funkcji przynależności zbiorów to
        4.3.1. Wyznacz punkt przecięcia
        4.3.2. Wykonaj operację na poziomach przynależności w punkcie przecięcia
               i dodaj nowy punkt do opisu wyniku
    4.4. Wykonaj operację na poziomach przynależności i dodaj punkt do opisu
         wyniku
```

7. Agregacja zbiorów rozmytych

Operacja agregacji zbiorów rozmytych pozwala na połączenie wniosków wygenerowanych przez różne reguły dla tych samych zmiennych wyjściowych [5-9]. Agregacja może zostać wykonana przez przecięcie lub sumę zbiorów w zależności od przyjętego modelu wnioskowania[5]. W tym wypadku problem sprowadza się do wykonania N-1 operacji przecięcia lub sumy dla N agregowanych zbiorów, wykorzystując algorytm przedstawiony w poprzednim punkcie.

Do agregacji można również użyć operatorów wyznaczania średniej [5,8,9]. W tym wypadku dla każdego elementu należy obliczyć jego średni poziom przynależności na podstawie wszystkich zbiorów wejściowych. Podejście to wymaga rozszerzenia algorytmu obliczania przecięcia i sumy, tak aby dla każdego kolejno rozważanego elementu wyznaczać wartości funkcji przynależności wszystkich zbiorów wejściowych:

```
1. Pobierz pierwszy punkt z lewej spośród opisu wszystkich zbiorów
2. Oblicz wartość funkcji przynależności dla reszty zbiorów
3. Wykonaj operację średniej na wszystkich poziomach przynależności i dodaj
   punkt do opisu wyniku
4. Dopóki istnieją nie przetworzone punkty opisu jakiegokolwiek zbioru wykonuj :
    4.1. Pobierz kolejny punkt opisu
    4.2. Oblicz wartość funkcji przynależności dla reszty zbiorów
    4.3. Jeśli nastąpiły przecięcia funkcji przynależności to
        4.3.1. Wyznacz punkty przecięcia
        4.3.2. Wykonaj operację średniej na poziomach przynależności punktów
               przecięcia i dodaj nowy punkt do opisu wyniku
    4.4. Wykonaj operację średniej na poziomach przynależności i dodaj nowy
         punkt do opisu wyniku
```

Powyższy algorytm nie został zaimplementowany ze względu na dużo większe skomplikowanie spowodowane koniecznością przetwarzania wielu zbiorów rozmytych, szczególnie

[5] przecięcie – model logiczny, suma – model koniunkcyjny (system Mamdaniego)

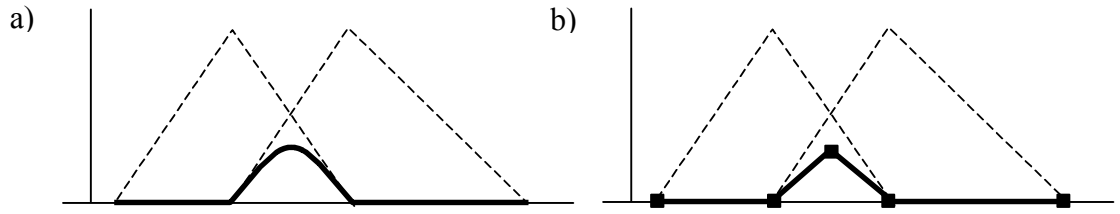
wyznaczania przecięć wszystkich funkcji. Ponadto, podejście nie pozwala na pewne uproszczenia, które zostały omówione w dyskusji na temat błędów w punkcie 8.

Omawiana biblioteka pozwala jednak na realizację agregacji z użyciem operatora średniej arytmetycznej, opierając się na wcześniejszym algorytmie pracującym z dwoma zbiorami. Do tego celu została zdefiniowana prosta operacja sumy dwóch wartości, którą podaje się algorytmowi do wykonania. W ten sposób po przetworzeniu dwóch zbiorów otrzymuje się funkcję przynależności odpowiadającą sumie dwóch funkcji wejściowych. Przetwarzając w ten sposób kolejne zbiory, uzyskuje się na wyjściu funkcję przynależności, będącą sumą wszystkich podanych funkcji. Ostatnim krokiem jest przetworzenie zbioru wyjściowego dzieląc każdą składową y wszystkich jego punktów opisu przez liczbę zbiorów biorących udział w agregacji. Tym samym uzyskuje się w każdym punkcie średnią arytmetyczną.

Zaprezentowany sposób przetwarzania wielu zbiorów z opisem odcinkowo-liniowym ma dużą zaletę. Pozwala mianowicie na bieżąco usuwać zbędne punkty opisu zbioru znacznie zmniejszając złożoność obliczeniową kolejnych agregacji cząstkowych. Problem został dokładniej opisany w punkcie 8.2.

8. Błędy przetwarzania związane z odcinkowo-liniową reprezentacją funkcji przynależności[6]

Przedstawiony algorytm, wykonujący operację na dwóch zbiorach rozmytych, daje dokładne wyniki jedynie w przypadku zastosowania przecięcia i sumy realizowanych przez odpowiednio t-normę minimum oraz s-normę maksimum. Przy zastosowaniu dowolnych s-norm i t-norm otrzymuje się jedynie liniowe przybliżenia pomiędzy wyznaczonymi dokładnie punktami opisu funkcji przynależności, ponieważ dokładny wynik jest nieliniowy. Na rys. 8 przedstawiono wynik operacji przecięcia zbiorów z użyciem iloczynu jako t-normy.



Rys. 8. Przecięcie zbiorów rozmytych z użyciem iloczynu jako t-normy: a) wynik dokładny, b) wynik rozważanego algorytmu

Fig. 8. Intersection of fuzzy sets with product t-norm: a) precise result, b) result of considered algorithm

[6] Rozważania w tym punkcie nie poruszają oczywistych błędów odcinkowo-liniowej postaci funkcji oraz reprezentacji liczb zmiennoprzecinkowych, a dotyczą wyłącznie błędów wnoszonych przez zaprezentowane algorytmy

Stosując odcinkowo-liniową reprezentację funkcji przynależności, jedynym rozwiązaniem jest zwiększenie liczby próbkowanych wartości pomiędzy punktami opisu, zwiększając tym samym opis zbioru wyjściowego.

Zmodyfikowany algorytm realizujący operację na dwóch zbiorach mógłby przed dodaniem każdego nowego punktu sprawdzać poprawność pozycji pośrednich i dodawać dodatkowe punkty w tym przedziale, gdy opis generowałby błąd większy od zadanego progu.

Zwiększenie dokładności obliczeń można również uzyskać przez zagęszczenie istniejącego opisu zbiorów dodając dodatkowe punkty przed wykonaniem jakichkolwiek operacji. Zadanie to może być wykonane jednorazowo w czasie przygotowania systemu do pracy. Zaletą tego podejścia jest możliwość wykorzystania zaprezentowanych wcześniej algorytmów bez konieczności wprowadzania zmian.

8.1. Zagęszczenie opisu odcinkowo-liniowej funkcji przynależności

Omawiana biblioteka zawiera dwa algorytmy zwiększania zagęszczenia opisu. Pierwszy opiera się na określeniu minimalnej różnicy wartości funkcji przynależności pomiędzy dwoma sąsiednimi punktami opisu. Gdy różnica wartości *dy* pomiędzy dwoma sąsiednimi punktami jest większa od zadanego progu minimalnego, algorytm dodaje odpowiednią ilość nowych punktów opisu w stałych odstępach.

Poniżej zamieszczono rozważany algorytm, w którym funkcja `dodaj_punkt` odpowiada dodaniu kolejnego punktu do nowego, zagęszczonego opisu zbioru. Funkcje `fabs` oraz `ceil` służą odpowiednio do obliczania wartości bezwzględnej i zaokrąglenia w górę. Natomiast funkcja `oblicz_przynaleznosc` wyznacza wartość funkcji przynależności pracując zgodnie z algorytmem przedstawionym w punkcie 3.

```
double dy,dx,dzielnik,x;  //zmienne pomocnicze
int i;                    //licznik

  //dodaj pierwszy punkt opisu
dodaj_punkt( punkty[0].x, punkty[0].y );

  //pętla po wszystkich punktach opisu
for(i=0; i<(size-1); i++){
  dy = fabs(punkty[i+1].y - punkty[i].y);    //oblicz dy
  if (dy > mindy){                           //jeśli podział za duży to :
    dzielnik = ceil( dy/mindy );               //oblicz podział dy
    dx = (pts[i+1].x - pts[i].x)/dzielnik;     //oblicz wyjściowy dx
    x = pts[i].x;
    while( (x+=dx) < pts[i+1].x ) {          //dodaj punkty
      dodaj_punkt(x, oblicz_przynaleznosc(x) );
    }
  }
    //dodaj punkt zamykający przedział
  dodaj_punkt( punkty[i+1].x, punkty[i+1].y );
}
```

W rzeczywistej implementacji dodawanie kolejnych punktów odbywa się z użyciem dynamicznej listy punktów, ponieważ na początku nie jest znana ilość potrzebnej pamięci na wynikowy opis zagęszczony. Ostatnim krokiem jest utworzenie nowego opisu funkcji przynależności na podstawie zebranych punktów.

Drugi zaimplementowany algorytm zwiększa zagęszczenie opisu, dzieląc każdy przedział na podaną, stałą ilość podprzedziałów. Tego typu podejście jest przydatne do prostego zwielokrotnienia opisu, szczególnie w przypadku funkcji przynależności ze wspomnianym wcześniej równomiernym podziałem jak na rys. 2.

Należy zauważyć, iż oba przedstawione podejścia nie zwiększają zagęszczenia przedziałów płaskich, których *dy = 0*.

```
//zmienna 'podzial' określa ilość podprzedziałów
//zmienna 'rozmiar' określa ilość punktów opisujących zbiór

double dx,step,x; //zmienne pomocnicze
int i,j;          //liczniki

dodaj_punkt( punkty[0].x, punkty[0].y );

  //pętla po wszystkich punktach opisu
for(i=0; i<(rozmiar-1); i++){
  if (punkty[i+1].y != punkty[i].y){   //nie dziel przedziałów płaskich
    dx = punkty[i+1].x - punkty[i].x;
    krok = dx/podzial;
    x = punkty[i].x;
    for (j = 1; j<podzial; j++ ){
       dodaj_punkt(x, oblicz_przynaleznosc(x) );
       x += step;
    }
  }
}
```

Zaprezentowane algorytmy pozwalają w prosty sposób dowolnie zwiększyć precyzję obliczeń do poziomu ograniczonego jedynie rozdzielczością liczb zmiennoprzecinkowych. Chcąc jednak uzyskać bardzo dużą dokładność, należy liczyć się ze stosunkowo dużym narzutem czasowym w porównaniu z systemem używającym funkcji mniej zagęszczonych. Spowodowane jest to liniową złożonością obliczeniową, którą charakteryzuje się większość algorytmów przetwarzających tak zdefiniowane zbiory rozmyte.

W tym miejscu należy zauważyć, iż stosowanie t-normy minimum oraz s-normy maksimum nie wymaga zagęszczania, pozwalając na uzyskanie dokładnych wyników przy minimalnej postaci opisu zbiorów[7]. Systemy wykorzystujące operacje tego typu, stworzone za pomocą omawianej biblioteki, będą działały najszybciej przy zachowaniu największej dokładności.

[7] Sytuację wyraźnie obrazuje rys. 7 w punkcie 6

8.2. Kompresja opisu odcinkowo-liniowej funkcji przynależności

Podczas wykonywania operacji sumy lub przecięcia na zbiorach z użyciem opisywanych algorytmów bardzo często powstaje nadmiarowy opis płaskich segmentów funkcji przynależności, które nie przyczyniają się do zwiększenia precyzji. Przypadek ten dobrze ilustruje przedstawiony w rozdziale 6. rys. 7, który wyraźnie obrazuje nadmiarowe punkty na poziomie 0 po przecięciu zbiorów oraz na poziomie 1 po operacji sumy.

Gdy zbiory posiadają bardziej zagęszczony opis, ilość nadmiarowych punktów na płaskich odcinkach może być bardzo duża. Oczywiście zwiększa to czas pracy całego projektowanego systemu ze względu na konieczność przetwarzania zbędnych punktów.

Rozwiązaniem jest dodawanie do wynikowego opisu podczas wykonywania operacji tylko tych punktów, które definiują zmienny poziom wartości funkcji. Wymaga to nieznacznej modyfikacji algorytmu operacji w kroku samego dodania nowego punktu.

Proces usuwania nadmiarowych informacji można też wykonać niezależnie. Wystarczy przejrzeć kolejne punkty opisu funkcji przynależności i usunąć niepotrzebne. Proces został nazwany kompresją opisu funkcji i realizuje go następujący algorytm:

```
//zmienna 'rozmiar' określa ilość punktów opisujących zbiór

  //zmienne pomocnicze
int i, pozycja;        //licznik oraz pozycja ostatniego elementu
double ost_y;          //wartość y ostatnio przetworzonego punktu
int nowy_rozmiar;      //nowy rozmiar

pozycja = 1;           //inicjalizacja
ost_y = punkty[0].y;
nowy_rozmiar = rozmiar;

  //pętla po wszystkich punktach opisu
for(i=1; i<rozmiar-1; i++){
  if ( ost_y == punkty[i].y && ost_y == punkty[i+1].y){
      nowy_rozmiar--;  //jeśli punkt "i" jest nadmiarowy to zmniejsz rozmiar
  } else {
      pozycja++;       //jeśli punkt "i" nie jest nadmiarowy to kontynuuj
  }
  ost_y = punkty[i].y;
  punkty[pozycja] = punkty[i+1]; //skopiuj ostatni punkt na aktualny koniec
}

  //skopiuj ostatni punkt
punkty[pozycja].x = punkty[i].x;
punkty[pozycja].y = punkty[i].y;

rozmiar = nowy_rozmiar;
```

Usuwanie nadmiarowych punktów po każdej operacji znacznie przyspiesza pracę systemu, szczególnie w przypadku agregacji wielu zbiorów rozmytych. Przy zwiększonym zagęszczeniu opisów efekt jest jeszcze bardziej widoczny, ponieważ kolejne kroki agregacji nie kumulują dużej ilości niepotrzebnych informacji.

9. Podsumowanie

Podstawowymi zaletami przedstawionego w artykule podejścia opisu zbiorów rozmytych jest prosty format reprezentacji, charakteryzujący się dużą elastycznością. Systemy rozmyte oparte na odcinkowo-liniowych funkcjach pozwalają testować i wdrażać dowolne formy opisu funkcji przynależności bez konieczności modyfikacji formatu danych. Ta cecha charakteryzuje również algorytmy pracujące z tego typu zbiorami, ponieważ bazują one na formacie zapisu, który zawsze pozostaje odcinkowo-liniowy. Dlatego stosowanie różnie opisanych funkcji nie pociąga za sobą konieczności modyfikacji raz wprowadzonych algorytmów.

Duża elastyczność wiąże się oczywiście z największą wadą podejścia, jaką jest stosunkowo duży nakład obliczeniowy oraz większe zużycie pamięci przy większym zagęszczeniu opisu funkcji. Dzieje się tak, ponieważ prawie wszystkie przedstawione algorytmy cechują się liniową złożonością obliczeniową zależną od rozmiaru opisu. Należy jednak zauważyć, iż zadanie stworzenia uniwersalnej biblioteki pozwalającej na tworzenie systemów rozmytych zakładających dowolną postać opisu funkcji przynależności oraz pozwalających na wybór spośród wielu dostępnych operacji jest zadaniem bardzo trudnym. Dlatego dobrze znany problem znalezienia odpowiedniego stosunku pomiędzy dokładnością obliczeń a szybkością działania pozostawia się użytkownikowi[8].

Kolejne prace nad biblioteką skupiają się wokół stworzenia gotowych modułów (klas programistycznych), reprezentujących pełne regułowe systemy wnioskowania. Ich zaletą będzie bardzo proste zarządzanie, co niebywale ułatwi pracę użytkownikom z niewielkim doświadczeniem. Ponadto, będą one na tyle elastyczne, iż pozwolą na szeroką konfigurację doświadczonym specjalistom.

W dalszej perspektywie będą wprowadzane moduły, pozwalające na tworzenie systemów opartych na zbiorach rozmytych typu drugiego. Początkowo również w podejściu opartym na reprezentacji odcinkowo-linowej, a później na mniej elastycznym lecz szybszym rozwiązaniu.

LITERATURA

1. Zadeh L.A.: Fuzzy Sets. Information and Control, 8, s. 338-353, 1965.
2. Zimmermann H.-J.: Fuzzy set theory and it's applications. Kluwer-Nijhoff, Boston 1985.
3. Klir G.J., Folger T.A.: Fuzzy Sets, Uncertanity, and Information. Prentice-Hall, Englewood Cliffs 1988.

[8] Dotyczy to ogólnego przypadku ponieważ przy wykorzystaniu w systemie t-normy minimum oraz s-normy maksimum podejście działa szybko i dokładnie

4. Klir G.J., Yuan B.: Fuzzy Sets and Fuzzy Logic. Theory and Applications. Prentice-Hall, Upper Saddle River 1995.
5. Łęski J.: Systemy neuronowo-rozmyte. WNT, Warszawa 2008.
6. Rutkowski L.: Metody i techniki sztucznej inteligencji. PWN, Warszawa 2006.
7. Rutkowska D., Piliński M., Rutkowski L.: Sieci neuronowe, algorytmy genetyczne i systemy rozmyte. PWN, Warszawa 1999.
8. Wang L.-X.:A course in fuzzy systems and control. Prentice-Hall, New Yourk 1998.
9. Yager R.R.: On mean type aggregation. IEEE Trans. Systems, Man and Cybernetics – Part B: Cybernetics, 26(2), s. 209-221, 1996.

Recenzent: Prof. dr hab. inż. Jacek Łęski

Wpłynęło do Redakcji 12 czerwca 2008 r.

Abstract

Paper describes basic algorithms of a library providing easy to approach development of fuzzy systems. First versions are implemented in C++ language. A goal of the project is to use other popular object oriented programming languages like Java and C#.

A special feature of presented approach is piecewise linear description of fuzzy set's membership function which provides a very flexible solution. Membership function is represented with vector storing subsequent points of it's description. In programming language it is represented with an array of a simple point structures that consists of two elements x and y.

Less complex operations are described at the beginning. First operation calculates a membership value for given element. Two algorithms are presented. One for general purposes and second, faster for dedicated solutions.

Algorithm of fuzzyfication is described next. Solution is based on configuring a fuzzyfying set which is shifted with a numerical value that creates the result.

Further, paper provides precise description of algorithms for defuzzyfication process. Four popular methods are implemented. First is known as maximum method that returns an element with the highest membership. Other three are based on center of gravity method[9]. Basic COG method algorithm is described first. Modifications are considered next to present

[9] COG – Center Of Gravity

indexed version ICOG and modified indexed version MICOG. Two last methods allow to omit those values of a membership function which are below given level.

Subsequent algorithms consider intersection, union and aggregation of sets. One common approach is described with analysis of different use cases.

The last part concerns errors produced with presented approach and provides method of their reduction. Solution increases density of set description which allows to define any precision but with cost of calculation time.

The greatest advantage of presented library is flexibility that allows to create complex fuzzy systems based on very simple data form. Author's goal is to develop and extend the tool. Next modules will provide very easy to use fuzzy rule-based systems for different applications.

Adres

Przemysław KUDŁACIK: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, przemyslaw.kudlacik@polsl.pl