Procesarea imaginilor folosind logica fuzzy

M. Ivanovici

6 noiembrie 2007

BREVIAR TEORETIC

1 Conceptele de bază ale logicii fuzzy

Logica clasică sau booleană este construită pe două valori de adevăr: "adevărat" sau "fals" (TRUE sau FALSE, 1 sau 0): un obiect este fie "alb" fie "negru", fie "mare" fie "mic", neexistând o situaţie intermediară. Logica fuzzy, formulată de către Zadeh în 1965, oferă o modalitate matematică de a cuantifica incertitudinea unei exprimări vagi sau incomplete folosite pentru transmiterea unei anumite informaţii: "e cam mare", "e aproape negru", "e 75% adevărat".

Diferenţa esenţială dintre logica clasică şi cea fuzzy, este faptul că legea "terţului exclus", definită de Aristotel, nu mai este valabilă. În Figura 1 este ilustrată diferenţa între logica booleană şi cea fuzzy—dacă în primul caz există o delimitare clară între cele două valori de adevăr, în al doilea caz există o zonă de nedeterminare, în care valoarea de adevăr poate fi 0 şi 1 în acelaşi timp, într-o anumită măsură dată de gradul de apartenenţă.

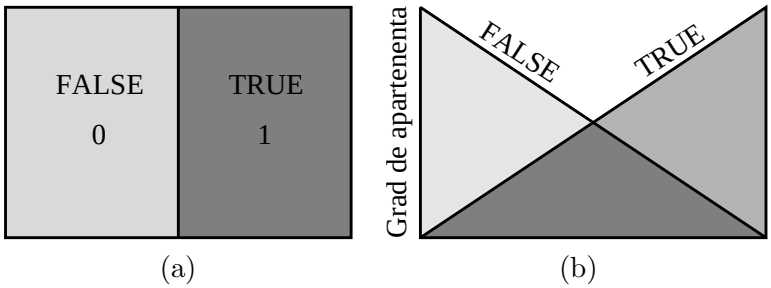


Figure 1: Ilustrarea mulţimilor "TRUE" şi "FALSE" în (a) logica booleană şi (b) logica fuzzy.

Logica fuzzy poate fi folosită atunci când nu există o delimitare clară între mulţimile ce reprezintă valorile unei mărimi sau proprietăţi, sau evenimentele sau stările posibile ale unei variabile aleatoare.

1.1 Mulţimile fuzzy

Conceptul de mulţime fuzzy, introdus de către Zadeh, a apărut ca o urmare firească a imposibilităţii de a modela un sistem indefinit (engl. ill-defined) cu ajutorul unor instrumente matematice precise, cum ar fi cele ale teoriei probabilistice.

Fie X o colecţie de obiecte. Mulţimea fuzzy A definită pe mulţimea X este o mulţime de perechi ordonate:

$$A = \{\mu_A(x)/x\} \tag{1}$$

unde $\mu_A(x)$ reprezintă *funcţia caracteristică* sau *funcţia de apartenenţă* a lui x la mulţimea A. Semnul "/" este folosit doar pentru a delimita valoarea reală x de valoarea funcţiei de apartenenţă $\mu_A(x)$.

Funţia de apartenenţă se defineşte pe intervalul de valori posibile ale lui x cu valori reale, în intervalul [0,1]: $\mu_A \rightarrow [0, 1]$.

Dacă A conţine doar 0 şi 1, atunci A nu mai este o mulţime fuzzy, ci una caracteristică logicii clasice. Dacă funcţia de apartenenţă este discretă, atunci mulţimea A se scrie ca:

$$A = \mu_1/x_1 + \mu_2/x_2 + ... + \mu_n/x_n = \sum_{i=1}^{n} \mu_i/x_i \tag{2}$$

Dacă funcţia de apartenenţă este o funcţie continuă, atunci mulţimea fuzzy A se scrie ca:

$$A = \int_X \mu_A(x)/x \tag{3}$$

unde prin integrare se înţelege totalitatea punctelor $\mu_A(x)/x$.

1.2 Proprietăţile mulţimilor fuzzy

- **Normalitatea**. O mulţime fuzzy este *normală* dacă valoarea maximă a funcţiei sale de apartenenţă este 1:

  $$\vee_x \mu(x) = 1$$

  unde $\vee_x$ reprezintă supremul sau maximul lui $\mu(x)$. În caz contrar, mulţimea se zice că este *subnormală*. Figura 2 prezintă două astfel de mulţimi fuzzy.

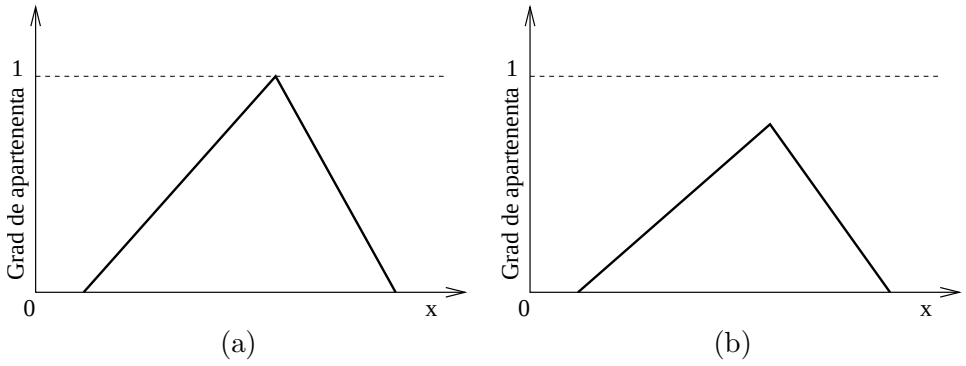


Figure 2: O mulţime fuzzy (a) normală şi (b) subnormală.

- **Convexitatea**. O mulţime fuzzy A este convexă dacă şi numai dacă mulţimile A_α definite ca:

$$A_\alpha = \{x | \mu_A(x) \geq \alpha\}$$

sunt convexe pentru orice $\alpha \in [0, 1]$. În Figura 3 sunt exemplificate două astfel de mulţimi. Mulţimea A_α conţine valorile pentru care funcţia de apartenenţă este egală sau mai mare decât pragul α.

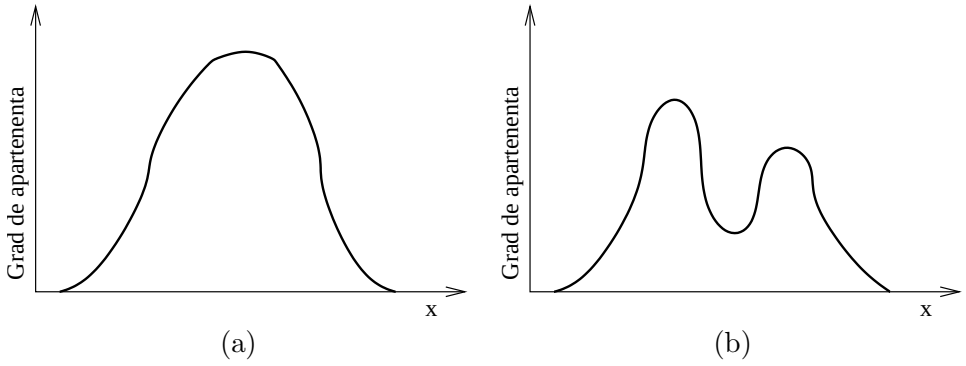


Figure 3: O mulţime fuzzy (a) convexă şi (b) neconvexă.

- **Punctul "crossover"**. Punctul "crossover" al unei mulţimi fuzzy A este elementul pentru care funcţia de apartenenţă are valoarea 0.5.
- **Valori singulare fuzzy**. O valoare singulară (engl. singleton) este o mulţime fuzzy care are un grad de apartenenţă pentru o singură valoare. Fie A o valoare singulară pentru un interval X, $x \in X$, atunci A poate fi scrisă ca:

$$A = \mu / x$$

Folosind această definiţie, o mulţime fuzzy poate fi considerată o reuniune de valori singulare.

1.3 Operaţii cu mulţimi fuzzy

- **Intersecţia fuzzy**. Interseţia a două mulţimi fuzzy este echivalentă cu operaţia "ŞI logic", reprezentând minimul dintre două grade de apartenenţă:

$$A(x) \cap B(x) = \sum \mu_A(x) \wedge \mu_B(x)$$

unde $\wedge$ reprezintă funcţia minim.

Rezultatul operaţiei de interseţie fuzzy dintre mulţimile A şi B din Figura 4 este reprezentat de către aria haşurată.

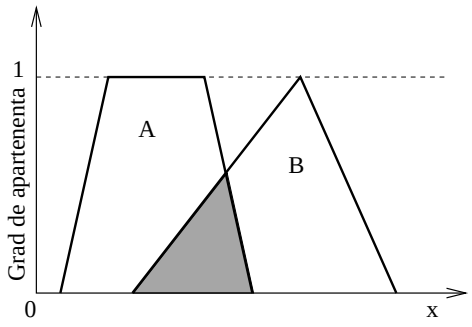


Figure 4: Interseţia a două mulţimi fuzzy.

- **Reuniunea fuzzy**. Reuniunea a două mulţimi fuzzy este interpretată ca funcţia "SAU logic", iar pentru două grade de apartenenţă, $\mu_A(x)$ şi $\mu_B(x)$,va lua valoarea maximă:

$$A(x) \cup B(x) = \sum \mu_A(x) \vee \mu_B(x)$$

unde $\vee$ reprezintă funcţia maxim. În Figura 5 este figurat rezultatul operaţiei de reuniune fuzzy.

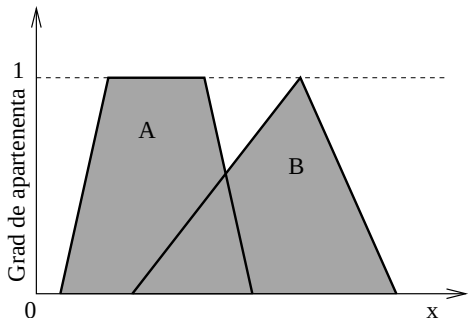


Figure 5: Reuniunea a două mulţimi fuzzy.

- **Complementul fuzzy**. Complementul unei mulţimi fuzzy A (vezi Figura 6, echivalent cu operaţia de negare logică, este definit de:

$$\overline{A} = \sum 1 - \mu_A(x)$$

unde cu $\overline{A}$ am notat complementul fuzzy al mulţimii A.

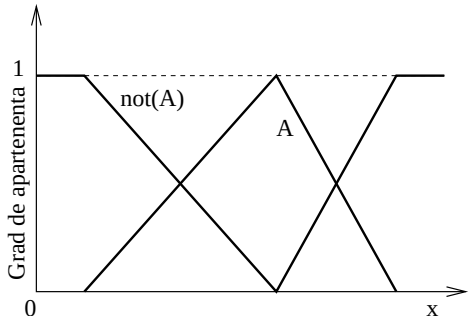


Figure 6: Complementul unei mulţimi fuzzy.

- **Combinaţia convexă**. Combinaţia convexă este un operator care combină mai multe mulţimi fuzzy într-o singură mulţime fuzzy, folosind ponderi asociate fiecărei mulţimi. Funcţia de apartenenţă totală $\mu_T(x)$ reprezintă combinaţia liniară a funcţiilor de apartenenţă $\mu_{A_1}, ..., \mu_{A_n}$:

$$\mu_T(x) = w_1(x)\mu_{A_1}(x) + w_2(x)\mu_{A_2}(x) + ... + w_n(x)\mu_{A_n}(x)$$

unde $w_1, w_2, ..., w_n$ sunt ponderile asociate mulţimilor fuzzy $A_1, A_2, ..., A_n$, astfel încât:

$$w_1(x) + w_2(x) + ... + w_n(x) = 1$$

- **Concentrarea fuzzy**. Concentrarea unei mulţimi fuzzy are ca rezultat o reducere a gradului de apartenenţă, prin ridicare la pătrat a valorilor funcţiei de apartenenţă. Daca A este o mulţime fuzzy:

$$A = \{\mu_1/x_1 + \mu_2/x_2 + ... + \mu_n/x_n\}$$

atunci concentratorul fuzzy aplicat acestei mulţimi se notează cu $CON(A)$ şi este definit astfel:

$$CON(A) = A^2 = \{\mu_1^2/x_1 + \mu_2^2/x_2 + ... + \mu_n^2/x_n\}$$

- **Dilatarea fuzzy**. Dilatarea fuzzy este un operator care măreşte gradul de apartenenţă a unei valori la mulţimea fuzzy, prin aplicarea operatorului de extragere a rădăcinii pătrate. Operaţia de dilatare se notează cu $DIL(A)$ şi se defineşte ca:

$$DIL(A) = A^{0.5} = \{\mu_1^{0.5}/x_1 + \mu_2^{0.5}/x_2 + ... + \mu_n^{0.5}/x_n\}$$

Este evident faptul că operaţia de dilatare are un efect opus celei de concentrare.

- **Plus şi minus fuzzy**. Operaţiile de plus şi minus fuzzy au un efect similar cu operaţiile de concentrare şi dilatare, cu deosebirea că efectul este mai puţin pronunţat. Pentru o mulţime fuzzy A, aceste operaţii se definesc astfel:

$$Plus(A) = A^{1.25}$$
$$Minus(A) = A^{0.75}$$

- **Intensificarea fuzzy**. Acest operator măreşte gradul de apartenenţă pentru valorile mai mari decât punctul de "crossover" ($\alpha = 0.5$) şi îl micşorează pentru acele valori mai mici decât acest punct. Pentru o mulţime fuzzy A, $x \in A$, atunci intensificarea fuzzy se defineşte ca:

$$\mu_{INT(A)}(x) \geq \mu_A(x), \quad \mu_A(x) \geq 0.5$$
$$\mu_{INT(A)}(x) \leq \mu_A(x), \quad \mu_A(x) < 0.5$$

1.4 Funcţiile de apartenenţă

Pentru o anumită aplicaţie care foloseşte logica fuzzy, alegerea funcţiilor de apartenenţă reprezintă elementul cheie, de care vor depinde performanţele acelei aplicaţii. În spiritul logicii fuzzy, funcţiile de apartenenţă ar trebui să reflecte o măsură subiectivă, mai degrabă decât una obiectivă.

În unele cazuri, funcţiile de apartenenenţă sunt generate pe baza unor date experimentale sau de antrenare, folosind tehnici de "clustering" sau de clasificare. Un algoritm de "clustering" sau clasficare poate fi folosit pentru a estima distribuţiile datelor de intrare, pe baza cărora pot fi generate funcţiile de apartenenţă.

Alegerea funţiilor de apartenenţă depinde în general de problemă, dar există câteva funcţii folosite în mod uzual, de formă triunghiulară, trapezoidală, în formă de clopot sau în formă de "S".

Funcţia de apartenentă triunghiulară se notează cu $\Lambda(x; a, b, c)$ şi are următoarea expresie matematică:

$$\Lambda(x; a, b, c) = \begin{cases} 0 & x \leq a, \\ \frac{x-a}{b-a} & a < x \leq b, \\ \frac{c-x}{c-b} & b < x \leq c, \\ 0 & x > c. \end{cases} \tag{4}$$

Funcţia de formă trapezoidală se notează cu $\Pi(x; a, b, c, d)$ şi are expresia:

$$\Pi(x; a, b, c, d) = \begin{cases} 0 & x \leq a, \\ \frac{x-a}{b-a} & a < x \leq b, \\ 1 & b < x \leq c, \\ \frac{d-x}{d-c} & c < x \leq d, \\ 0 & x > d. \end{cases} \tag{5}$$

Funcţia în formă de "S" propusă de Zadeh se notează cu $S(x; a, b, c)$ şi este:

$$S(x; a, b, c) = \begin{cases} 0 & x \leq a, \\ 2(\frac{x-a}{c-a})^2 & a < x \leq b, \\ 1 - 2(\frac{x-c}{c-a})^2 & b < x \leq c, \\ 0 & x > c. \end{cases} \tag{6}$$

Funcţia de apartenenţă în formă de clopot $\pi(x; a, b, c)$, similară funcţiei Gauss, se poate defini pe baza funcţiei S astfel:

$$\pi(x; b, c) = \begin{cases} S(x; c-b, c-b/2, c) & x \leq c, \\ 1 - S(x; c, c+b/2, c+b) & x > c. \end{cases} \tag{7}$$

După alegerea funcţiilor de apartenenţă, de regulă este nevoie de a construi un set de funcţii de apartenenţă pentru toate mulţimile fuzzy asociate cazurilor problemei. De exemplu, în Figura 7 este prezentat un set de funcţii

triunghiulare şi trapezoidale uniform distribuite pe intervalele de nivele de gri corespunzătoare atributelor “foarte mic”, “mic”, “mediu”, “mare” şi “foarte mare”.

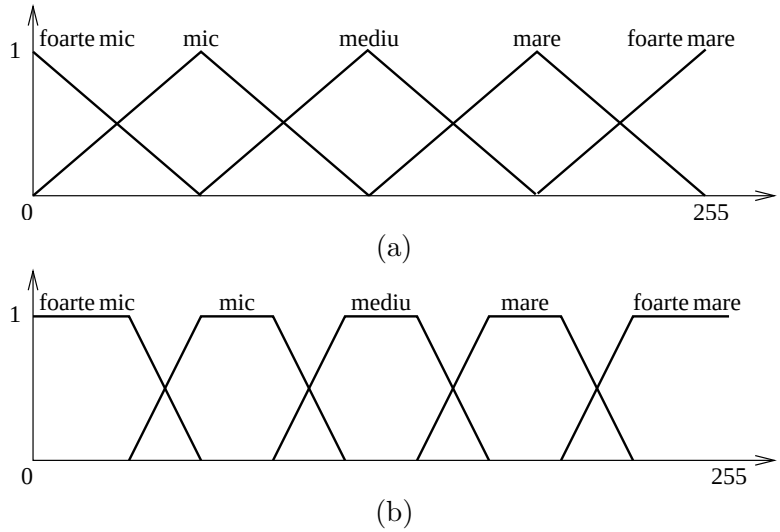


Figure 7: Set de funcţii de apartenenţă (a) triunghiulare şi (b) trapezoidale.

2 Detecţia contururilor folosind logica fuzzy

Datorită capacităţii abordărilor fuzzy de a reprezenta informaţiile imprecise sau neclare, acestea sunt mai puţin sensibile la alegerea parametrilor, cum ar fi valoarea de prag pentru un detector de contur.

Abordarea propusă în [2] combină reguli de tip *If-Then* pentru a atribui fiecărui pixel o valoare reprezentând gradul de apartenenţă a acestuia la mulţimea pixelilor de contur, formând astfel o mulţime a potenţialilor pixeli de contur[1].

Funcţii de apartenenţă caracterizează diferenţele dintre nivelul de gri al pixelului considerat la un moment dat şi nivelele de gri ale pixelilor vecini. Aceste diferenţe, în valoare absolută, pot fi caracterizate ca fiind “mici” (small) sau “mari” (large), astfel fiind definite două mulţimi fuzzy cărora le pot aparţine: mulţimea fuzzy a diferenţelor “mici” şi mulţimea fuzzy a diferenţelor “mari”.

Funcţiile de apartenenţă, μ_{small} şi μ_{large} sînt definite ca fiind complementare ($\mu_{large} = 1 - \mu_{small}$), şi au următoarele expresii matematice:

[1]PEP = (eng.) Potential Edge Pixel.

$$\mu_{small}(x) = \begin{cases} 1 & x \in [0,5], \\ \frac{128-x}{123} & x \in (5,128], \\ 0 & x \in (128,255]. \end{cases} \quad (8)$$

$$\mu_{large}(x) = \begin{cases} 0 & x \in [0,5], \\ \frac{x-5}{123} & x \in (5,128], \\ 1 & x \in (128,255]. \end{cases} \quad (9)$$

Graficul acestor funcţii de apartenenţă este prezentat în figura (8).

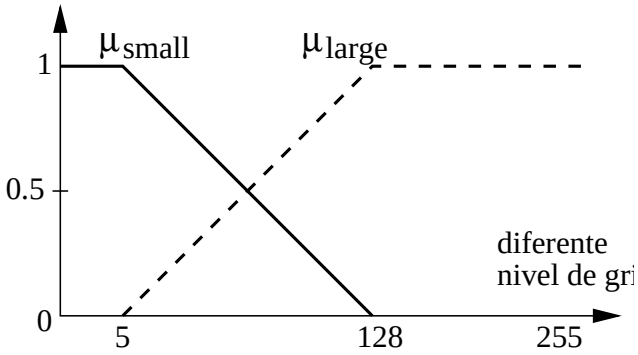


Figure 8: Funcţiile de apartenenţă ale diferenţelor de nivele de gri.

S-au folosit funcţii liniare pentru simplitate. Pragul inferior având valoarea 5 este ales diferit de zero pentru a evita o prea mare sensibilitate la zgomot.

Pentru aplicarea regulilor fuzzy *If-Then* se folosesc 8 măşti pătrate de dimensiune 3x3, astfel:

unde unii pixeli vecini pixelului curent (x) au fost marcaţi cu S, de la "small", iar alţii cu L, de la "large".

Pentru fiecare mască, se notează cu D_S diferenţa în modul dintre valoarea pixelului central x, şi valoarea unui pixel notat cu S, şi se notează cu D_L diferenţele în modul dintre valoarea aceluiaşi pixel central şi valoarea unui pixel notat cu L. Aceste diferenţe sînt caracterizate de cele două mulţimi fuzzy definite anterior. Funcţiile de apartenenţă sînt asociate acestor diferenţe.

Pentru fiecare mască se defineşte o regulă fuzzy de tip *If-Then*, astfel:

If {*toate diferenţele* D_S *sunt* **mici**} *AND*
{*toate diferenţele* D_S *sunt* **mari**} *Then*
*pixelul central al măştii este considerat un potenţial pixel de contur (***PEP***-Potential Edge Pixel).*

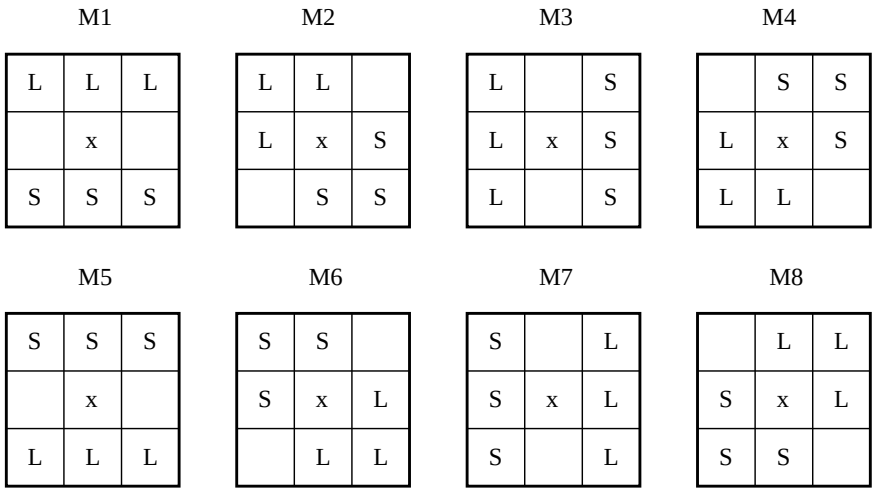


Figure 9: Măşti de tip compas.

Pentru fiecare mască M_i se calculează gradul de apartenenţă a pixelului central la mulţimea potenţialilor pixeli de contur, astfel:

$$\mu_{PEP\_M_i} = \top(toate\ \mu_{small}(D_S),\ toate\ \mu_{large}(D_L)) \tag{10}$$

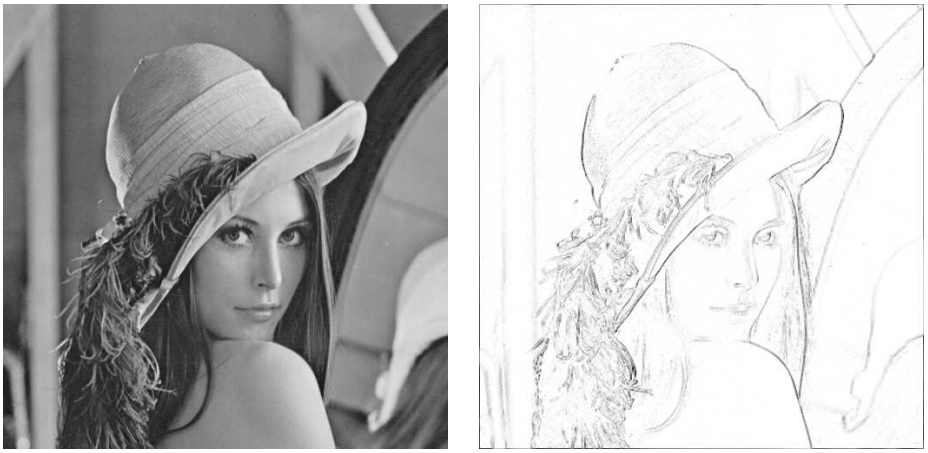
unde operatorul $\top$ reprezintă funcţia minim.

După ce au fost calculate cele 8 valori reprezentând gradele de apartenenţă la mulţimea fuzzy PEP, pentru fiecare mască în parte, se calculează pentru pixelul central gradul final de apartenenţă la mulţimea fuzzy PEP, astfel:

$$\mu_{PEP} = \bot_8(\mu_{PEP\_M_i}) \tag{11}$$

unde operatorul $\bot$ reprezintă funcţia de maxim.

Pentru fiecare pixel din imagine se calculează gradul μ_{PEP} de apartenenţă la mulţimea PEP şi i se atribuie o valoare de nivel de gri de $255 \cdot \mu_{PEP}$, care poate fi privită ca fiind valoarea în modul a unui operator de tip gradient. Direcţia acestui gradient poate fi direcţia dată de mască ce corespunde valorii maxime dintre cele 8 valori $\mu_{PEP\_M_i}$.



(a) (b)

Figure 10: Imaginea (a) originală şi (b) contururile obţinute folosind algoritmul fuzzy prezentat.

3 Segmentarea imaginilor folosind logica fuzzy

Algoritmii ce folosesc logica fuzzy pentru segmentarea imaginilor pot fi clasificaţi în următoarele categorii:

- **Grupare (clustering) fuzzy**. Această categorie de algoritmi reprezintă una din primele abordări a problemei segmentării imaginilor. Algoritmii de clustering pot fi folosiţi pentru a forma segmente, grupând pixeli ce respectă un anumit criteriu de similitudine. Funcţia de apartenenţă a pixelilor la o anumita clasă poate fi interpretată ca măsura compatibilităţii cu acel criteriu sau ideal. Din această categorie face parte algoritmul "fuzzy c-means" pe care îl vom prezenta în cele ce urmează.

- **Algoritmi bazaţi pe reguli**. Pentru segmentare pot fi folosite reguli de tip *If-Then*, pentru interpretarea conţinutului imaginii. Un exemplu de o astfel de regulă fuzzy este următorul:
  If pixelul este întunecat
  And vecinii lui sunt şi ei întunecaţi *And* omogeni
  Then pixelul aparţine fundalului.

- **Algoritmi bazaţi pe interpretarea fuzzy a informaţiei**. Folosind logica fuzzy putem defini entropia fuzzy sau divergenţa fuzzy, care pot fi folosite pentru a segmenta o imagine, de exemplu prin prăguirea histogramei. O funcţie de apartenenţă poate fi "glisată" de-alungul histogramei, iar pragul va fi ales astfel încât funcţia care defineşte entropia fuzzy să înregistreze un minim.

- **Geometrie fuzzy**. Măsuri geometrice fuzzy, cum ar fi indicele de acoperire al ariei sau indicele fuzzy al gradului în care un obiect este compact, pot fi folosite pentru a caracteriza diverse regiuni ale unei imagini. Minimizarea unor astfel de indici poate fi aplicată în cazul unor clasificări fuzzy.

Printre algoritmii de clustering amintim algoritmii *c-means*, cuantizare vectorială adaptivă[2] şi hărţi cu auto-organizare[3]. În cadrul acestei secţiuni despre segmentare, vom prezenta algoritmul de clustering *c-means* clasic şi varianta sa fuzzy.

Procesul de *clustering* reprezintă procesul de obţinere a unor partiţii (sau submulţimi) ale unei mulţimi de obiecte, folosind o anumită măsură a asemănării dintre obiectele ce alcătuiesc o partiţie. O astfel de măsură poate fi distanţa Euclidiană, în cazul unor vectori n-dimensionali. O partiţie a mulţimii poartă numele de *cluster*, iar centrele acestor partiţii poartă numele de *centroizi* sau *prototipuri*.

În continuare, vom folosi distanţa Euclidiană dintre doi vectori, X_i şi X_j, având componentele x_{ik} şi respectiv x_{jk}, ca fiind:

$$d(X_i, X_j) = \sqrt{\sum_k (x_{ik} - x_{jk})^2} \tag{12}$$

3.1 Algoritmul de clustering *c-means*

Algoritmul *c-means* reprezintă o metodă de învăţare nesupervizată, care poate fi folosită pentru clasificare atunci când numărul de clustere sau partiţii este cunoscut. Algoritmul Kohonen este un alt algoritm de învăţare nesupervizată.

Algoritmul *c-means* are următorii paşi:

1. Se alege numărul de partiţii, k,
2. Se aleg centroizii iniţiali ai partiţiilor, $c_1, c_2, ..., c_k$, ca fiind k vectori din setul de antrenare,
3. Se clasifică fiecare vector $x_l = [x_{l1}, x_{l2}, ..., x_{ln}]^T$, n fiind dimensiunea vectorilor setului de antrenare, ca fiind de tipul celui mai apropriat centroid c_i, folosind drept criteriu distanţa Euclidiană:

$$||x_l - c_i|| = \min_j ||x_l - c_j||$$

[2]Adaptive Vector Quantization (AVQ).
[3]Self-Organizing Map (SOM).

4. Se recalculează valorile estimate pentru centroizii c_i ai partiţiilor:

$$c_{im} = \frac{\sum_{X_{l_i} \in cluster\ i} x_{l_i m}}{N_i}$$

unde N_i este numărul de vectori din partiţia i,

5. Dacă nici unul din centroizii c_i nu se modifică la pasul precedent, atunci procesul de clustering încetează. Altfel se reia algoritmul de la pasul 3.

Pentru fiecare partiţie i rezultată se poate calcula varianţa valorilor din acea partiţie, folosind formula:

$$v_{im} = \sqrt{\frac{\sum_{X_{l_i} \in cluster\ i} (x_{l_i} m - c_{im})^2}{N_i}} \tag{13}$$

Având centrele si varianţele valorilor din fiecare partiţie, putem genera funcţii de apartenenţă pentru clasele sau partiţiile respective. Cu alte cuvinte, cu ajutorul algoritmului de clustering obţinem o împărţire a spaţiului în partiţii, pe baza cărora vom defini sau construi funcţii de apartenenţă pe care să le folosim apoi pentru o anumită aplicaţie ce foloseşte logica fuzzy, cum ar fi segmentarea. Generarea funcţiilor de apartenenţă pe baza centroizilor şi a varianţelor partiţiilor obţinute prinr-un algoritm de clustering este ilustrată în Figura 11, pentru cazul unor vectori bidimensionali.

3.2 Algoritmul de clustering *c-means* fuzzy

În varianta clasică a algoritmului *c-means* fiecare vector aparţine doar unei partiţii şi toate partiţiile sunt privite ca submulţimi disjuncte ale setului de vectori de antrenare. În practică există multe cazuri în care partiţiile nu sunt complet disjuncte, vectorii putând fi clasificaţi ca aparţinând mai multor partiţii în acelaşi timp. Într-un astfel de caz, separarea dintre partiţii devine o noţiune vagă, logica fuzzy fiind potrivită pentru abordarea unei astfel de probleme.

Algoritmul *fuzzy c-means* este cel mai cunoscut şi mai utilzat algoritm fuzzy de clustering. Algoritmul se bazează pe minimizarea iterativă a următoarei funcţii:

$$J(U, V) = \sum_{i=1}^{c} \sum_{k=1}^{n} u_{ik}^m |x_k - v_i|^2 \tag{14}$$

unde:

- $x_1, ..., x_n$ sunt vectori ai setului de antrenare,

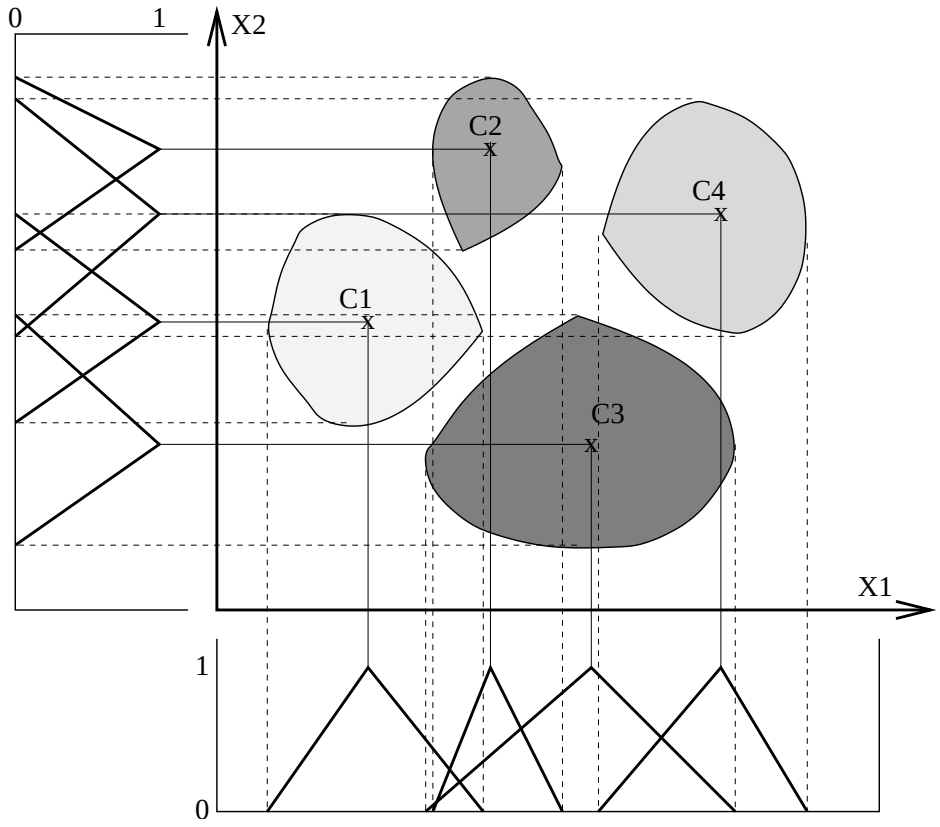


Figure 11: Generarea funcţiilor de apartenenţă pe baza centroizilor şi varianţelor partiţiilor.

- $V = v_1, ..., v_c$ sunt centroizii partiţiilor,
- $U = [u_{ik}]$ este o matrice de dimensiune $c \times n$, în care u_{ik} este valoarea de apartenenţă a vectorului x_k la partiţia i. Aceste valori de apartenenţă satisfac următoarele condiţii:

$$0 \leq u_{ik} \leq 1 \quad i = 1, 2, ..., c; \quad k = 1, 2, ..., n$$

$$\sum_{i=1}^{c} u_{ik} = 1 \quad k = 1, 2, ..., n$$

$$0 < \sum_{k=1}^{n} u_{ik} < n \quad i = 1, 2, ..., c$$

$m \in [1, \infty)$ este un factor de ponderare de tip exponent.

Funţia $J(U, V)$ reprezintă suma pătratelor distanţelor Euclidiene dintre fiecare vector şi centrul partiţiei corespunzătoare acelui vector, distanţe

ponderate de valorile de apartenenţă fuzzy. Pentru calculul centrului unei partiţii toţi vectorii sunt luaţi în considerare. Pentru fiecare vector, valoarea de apartenenţă la fiecare clasă depinde de distantą la centroidul corespunzător. Factorul de ponderare m reduce influenţa valorilor de apartenenţă mici. Cu cât m este mai mare, cu atât se reduce influenţa vectorilor cu valori de apartenenţă mici.

Algoritmul *fuzzy c-means* este unul iterativ şi are următorii paşi:

1. Se iniţializează $U^{(0)}$ cu valori aleatorii sau pe baza unei aproximări. Se iniţializează $V^{(0)}$ şi se calculează $U^{(0)}$. Se iniţializează numărul de iteraţii $\alpha = 1$. Se aleg numărul de partiţii c şi ponderea m.

2. Se calculează centroizii partiţiilor. Pentru $U^{(\alpha)}$ dat, se calculează $V^{(\alpha)}$ folosind formula:

$$v_i = \frac{1}{\sum_{k=1}^{n} u_{ik}^m} \sum_{k=1}^{n} u_{ik}^m x_{ik} \quad i = 1, 2, ..., c \tag{15}$$

3. Se actualizează valorile de apartenenţă. Pentru $V^{(\alpha)}$ dat se calculează $U^{(\alpha)}$ folosind formula:

$$u_{ik} = \frac{\left[\frac{1}{|x_k - v_i|^2}\right]^{\frac{1}{m-1}}}{\sum_{j=1}^{c} \left[\frac{1}{|x_k - v_j|^2}\right]^{\frac{1}{m-1}}} \quad i = 1, 2, ..., c;\ k = 1, 2, ..., n \tag{16}$$

4. Se opreşte iteraţia dacă:

$$max|u_{ik}^{(\alpha)} - u_{ik}^{(\alpha-1)}| \leq \epsilon \tag{17}$$

altfel se trece la iteraţia următoare, $\alpha = \alpha + 1$ şi se reia algoritmul de la pasul 2. ϵ este o valoare mică predefinită care reprezintă cea mai mică modificare acceptabilă pentru valorile lui U.

DESFĂŞURAREA LUCRĂRII

Problema 1. În continuare sunt prezentate implementările funcţiilor de apartenenţă μ_{small} şi μ_{large}:

```
double ImageViewer :: miu_small( int x )
{
  if( x >= 0 && x <= 5 )
    return 1;
  if( x > 5 && x < 128 )
    return 1. * ( 128 - x ) / 123;
}

double ImageViewer :: miu_large( int x )
{
  if( x > 128 && x <= 255 )
    return 1;
  if( x >= 5 && x <= 128 )
    return 1. * ( x - 5 ) / 123;
}
```

Problema 2. În continuare este prezentată implementarea în `C` a algoritmului de detecţie de contur folosind logica fuzzy prezentat. Citiţi şi înţelegeţi codul.

```
void ImageViewer :: fuzzy( void )
{
  int i, j,  w, h, k, l, m, n, p;
  int gri, niv, nivd;

  QRgb pixel, pixeld;

  QImage fuz;
  fuz = image.copy();

  int M[ 8 ][ 3 ][ 3 ]; //cele 8 masti de tip compas
  int dif[ 3 ][ 3 ];    //matricea diferentelor
  double miu[ 3 ][ 3 ]; //matricea miu( diferente )

  double min, max;
  double PEP, PEP_M[ 8 ];
  int nr;

  M[0][0][0] =  1;   M[0][0][1] =  1;   M[0][0][2] =  1;
  M[0][1][0] =  0;   M[0][1][1] =  0;   M[0][1][2] =  0;
  M[0][2][0] = -1;   M[0][2][1] = -1;   M[0][2][2] = -1;

  M[1][0][0] =  1;   M[1][0][1] =  1;   M[1][0][2] =  0;
  M[1][1][0] =  1;   M[1][1][1] =  0;   M[1][1][2] = -1;
```

```
M[1][2][0] =  0;   M[1][2][1] = -1;   M[1][2][2] = -1;

M[2][0][0] =  1;   M[2][0][1] =  0;   M[2][0][2] = -1;
M[2][1][0] =  1;   M[2][1][1] =  0;   M[2][1][2] = -1;
M[2][2][0] =  1;   M[2][2][1] =  0;   M[2][2][2] = -1;

M[3][0][0] =  0;   M[3][0][1] = -1;   M[3][0][2] = -1;
M[3][1][0] =  1;   M[3][1][1] =  0;   M[3][1][2] = -1;
M[3][2][0] =  1;   M[3][2][1] =  1;   M[3][2][2] =  0;

M[4][0][0] = -1;   M[4][0][1] = -1;   M[4][0][2] = -1;
M[4][1][0] =  0;   M[4][1][1] =  0;   M[4][1][2] =  0;
M[4][2][0] =  1;   M[4][2][1] =  1;   M[4][2][2] =  1;

M[5][0][0] = -1;   M[5][0][1] = -1;   M[5][0][2] =  0;
M[5][1][0] = -1;   M[5][1][1] =  0;   M[5][1][2] =  1;
M[5][2][0] =  0;   M[5][2][1] =  1;   M[5][2][2] =  1;

M[6][0][0] = -1;   M[6][0][1] =  0;   M[6][0][2] =  1;
M[6][1][0] = -1;   M[6][1][1] =  0;   M[6][1][2] =  1;
M[6][2][0] = -1;   M[6][2][1] =  0;   M[6][2][2] =  1;

M[7][0][0] =  0;   M[7][0][1] =  1;   M[7][0][2] =  1;
M[7][1][0] = -1;   M[7][1][1] =  0;   M[7][1][2] =  1;
M[7][2][0] = -1;   M[7][2][1] = -1;   M[7][2][2] =  0;

w = image.width();
h = image.height();

for( i = 1; i < w - 1 ; i++ )
  for ( j = 1; j < h - 1; j++ )
  {
    //pixelul curent (i,j)
    pixel = image.pixel( i, j );
    niv = qRed( pixel );

    //parcurge o vecinatate 3x3, pentru calculul diferentelor
    for( k = -1; k < 2; k++ )
      for( l = -1; l < 2; l++ )
      {
        pixeld = image.pixel( i + k, j + l );
        nivd = qRed( pixeld );

        //calculeaza diferenta intre pixelul considerat si
```

```
 //pixelii din vecinatatea sa ( 3x3 )
 dif[ k + 1 ][ l + 1 ] = (int)( abs(niv - nivd) );

 //verifica daca pixelul considerat respecta vreuna
 //din cele 8 reguli IF-THEN, calculind astfel PEP
for( m = 0; m < 8; m++ )
{
   //pentru fiecare regula utilizeaza o masca compas
   nr = 0;
   min = 2; //val > 1 (max)

  for( n = 0; n < 3; n++ )
    for( p = 0; p < 3; p++ )
    {
       switch( M[ m ][ n ][ p ] )
       {
         case -1:
           {
             miu[n][p] = miu_small( dif[n][p] );
             break;
           }
        case 0:
           {
             miu[n][p] = 0;
             break;
           }
         case 1:
           {
             miu[n][p] = miu_large( dif[n][p] );
             break;
           }
      }

     //numara de cite ori Ds / Dl sunt small / large
     if( miu[ n ][ p ] != 0 )
       nr++;

     //determina miu_ cel mai mic
     if( miu[ n ][ p ] < min && miu[ n ][ p ] != 0 )
       min = miu[ n ][ p ];
   }

  //daca toate Ds sint small si toate Dl sint large
  //adica avem 6 valori diferite de zero
```

```
            if( nr == 6 )
            {
              //pixelul considerat este un PEP
              PEP_M[ m ] = min;
            }
            else
              PEP_M[ m ] = 0;
          }

          //calculeaza PEP ca maximul valorilor PEP_M
          max = -1;

          for( m = 0; m < 8; m++ )
            if( PEP_M[ m ]  > max )
              max = PEP_M[ m ];

          if( max != 0 )
            PEP = max;
          else
            PEP = 0;

        }

        gri = 255 - (int)( 255 * PEP );
        fuz.setPixel( i, j, qRgb( gri, gri, gri ) );
    }

    image = fuz;
    pm = image;
    update();
}
```

Problema 3. Modificaţi funcţiile de apartenenţă μ_{small} şi μ_{large} şi observaţi efectele diverselor funcţii asupra rezultatului detecţiei de contur.

Bibliografie

[1] **Z. Chi, H. Yan, T. Pham**, *"Fuzzy Algorithms: With Applications to Image Processing and Pattern Recognition"*, Advances in Fuzzy Systems–Applications and Theory vol. 10, World Scientific, 1996.

[2] **T. Carron, P. Lambert**, *"Fuzzy Color Edge Extraction by Inference Rules–Quantitative Study and Evaluation of Performances"*, article.

[3] **Tizhoosh** *"Homepage of Fuzzy Image Processing"*, web page.