Threshold-Free Cluster Enhancement

Luigi Giugliano[1], Marco Mecchia[1]

[1]Università degli studi di Salerno

10 maggio 2016

Overview

Overview

Spatial information enhancing

- Tecnica che prevede l'utilizzo di informazioni spaziali per aumentare l'autenticità di estese aree di segnale.

- La motivazione risiede nel fatto che le regioni del segnale sono più estese del rumore e quindi trovare zone aumenta la possibilità che esse siano segnale vero e proprio e non rumore.

Cluster-based Thresholding

- L'approccio piú comune in neuroimaging.

- Problemi:
  - Necessitá di definire una soglia di clustering.
  - Sogliatura di tipo *hard*
  - Difficoltá nel riconoscimento di eventuali *subcluster*

Overview dell'algoritmo

- Procedura che tenta di superare i problemi degli approcci precedenti.
- Input: Un immagine statistica non processata.
- Output: Un immagine statistica in cui il valore di ogni voxel é un punteggio che rappresenta il contributo spaziale del cluster di cui fa parte.

- Clustering dell'immagine intrinseco.

Assegnazione dei punteggi(1/2)

Il punteggio del voxel p viene stabilito dalla seguente formula:

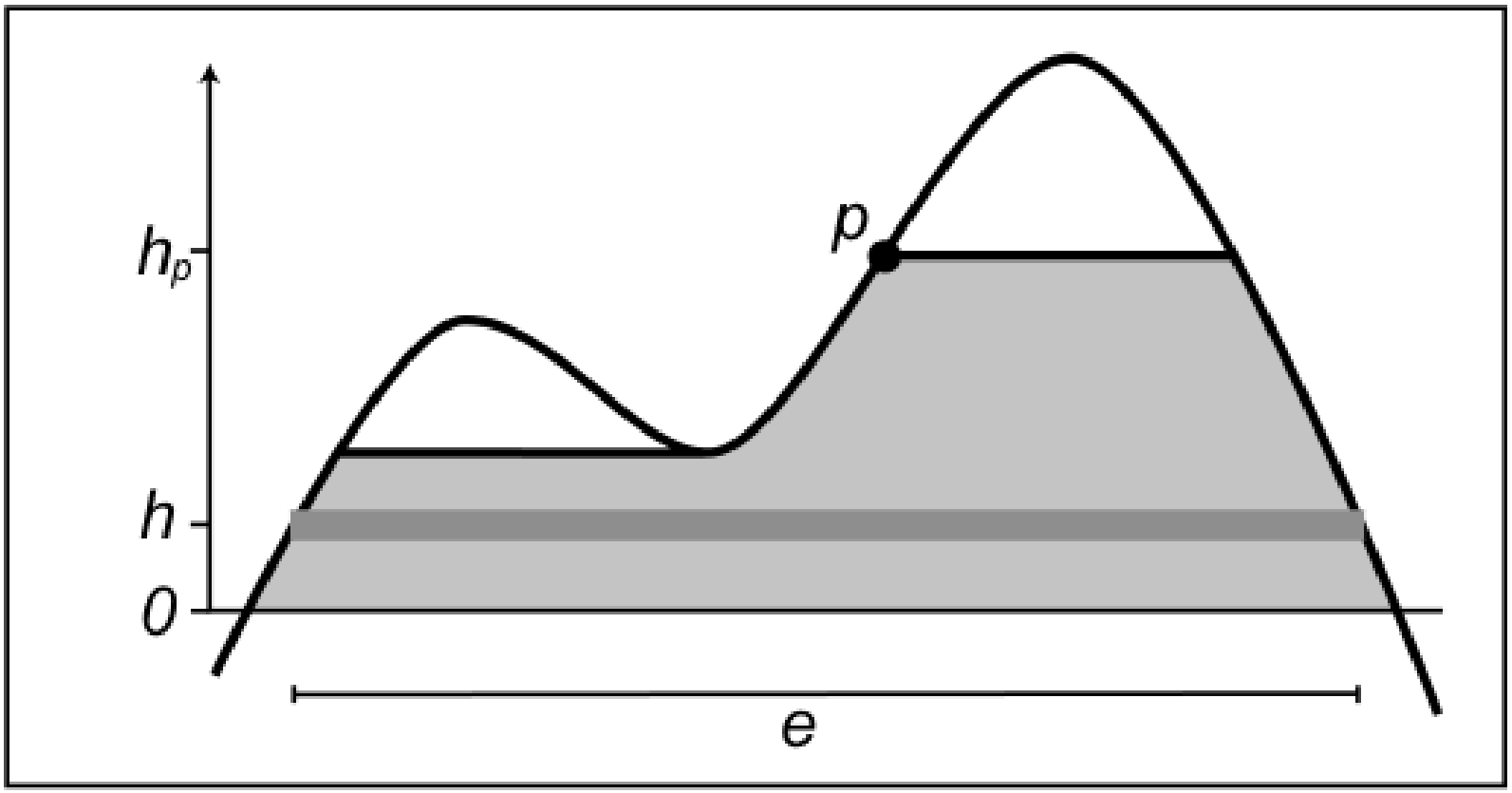
$$TFCE(p) = \int_{h=h_0}^{h_p} e(h)^E h^H dh$$

dove:

- h_p é il punteggio statistico del voxel p.
- $e(h)$ é l'area del cluster ad altezza h.
- E ed H sono costanti.

Questo integrale viene calcolato in pratica tramite una sommatoria ponendo $dh = 0.1$.

Assegnazione dei punteggi(2/2)



Overview

Suddivisione del codice

I file principali che compongono il plugin sono:

- Tfce.cpp
- Utilities.cpp

Tfce è il core del plugin, dove avviene il calcolo dei punteggi.

Utilities contiene tutte le funzioni di supporto.

Funzioni pubbliche (1/3)

L' unica funzione che viene esposta dal file **Tfce.h** è:

```
float * tfce_score(float * map, int dim_x, int dim_y,
    int dim_z, float E, float H, float dh);
```

Funzioni pubbliche (2/3)

Le funzioni che espone **Utilities.h** sono:

```
void findMinMax(float *map, int n, float *min, float
    *max, float * range);

int confront(float a, float b, char operation);

int * getBinaryVector(float * map, int n, int
    (*confront)(float, float), float value, int *
    numOfElementsMatching);
```

Funzioni pubbliche (3/3)

```
float * fromBinaryToRealVector(float * map, int n, int
    * binaryVector);

float * fill0(int n);

void apply_function(float * vector, int n, float (*
    operation) (float a, float b), float argument);

int linearIndexFromCoordinate(int x, int y, int z, int
    max_x, int max_y);

void coordinatesFromLinearIndex(int index, int max_x,
    int max_y, int * x, int * y, int * z);

float * copyAndConvertIntVector(int * vector, int n);
```

Dettagli della funzione tfce score

```
float * tfce_score(float * map, int dim_x, int dim_y,
    int dim_z, float E, float H, float dh){
__findMinMax(map, n, &minData, &maxData, &rangeData);
__precision = rangeData/dh;
__if (precision > 200) {
____increment = rangeData/200;
__} else{
____increment = rangeData/precision;
__}
__steps = ceil((maxData - minData) / (increment));
__#pragma omp parallel for
__for (i = 0; i < steps; i++) {
____ computeTfceIteration(minData + i*increment, map,
    n, dim_x, dim_y, dim_z, E, H, dh, toReturn);
__}
__return toReturn;
}
```

Funzione **computeTfceIteration**:

```
int * indexMatchingData = getBinaryVector(map, n,
    moreThan, h, &numOfElementsMatching);
clustered_map = find_clusters_3D(indexMatchingData,
    dim_x, dim_y, dim_z, n, &num_clusters);
extent_map = new int[n];
for (j = 0; j < n; ++j){
    extent_map[j] = 0;
}
delete [] indexMatchingData;
for (i = 1; i <= num_clusters; ++i) {
    numOfElementsMatching = 0;
    for (j = 0; j < n; ++j){
        if(clustered_map[j] == i){
            numOfElementsMatching++; }}
    for (j = 0; j < n; ++j) {
       if(clustered_map[j] == i)
           extent_map[j] = numOfElementsMatching; }}
```

```
clustered_map_float =
    copyAndConvertIntVector(extent_map, n);
apply_function(clustered_map_float, n, elevate, E);
apply_function(clustered_map_float, n, multiply,
    pow(h, H));
apply_function(clustered_map_float, n, multiply, dh);
for (i = 0; i < n; ++i) {
#pragma omp atomic
    toReturn[i] += (clustered_map_float[i]);
}
delete[] clustered_map_float;
delete[] clustered_map;
delete[] extent_map;
```

Funzione **getBinaryVector**:

```
int * getBinaryVector(float * map, int n, int
    (*confront)(float, float), float value, int *
    numOfElementsMatching){
    int * binaryVector = new int [n];
    (*numOfElementsMatching) = 0;
    int i;
    for (i = 0; i < n; ++i) {
        if(confront(map[i],value)){
            binaryVector[i] = 1;
            (*numOfElementsMatching)++;
        }
        else
            binaryVector[i] = 0;
        }
    return binaryVector;
}
```

Find connected components in binary images (1/2)

La funzione **find_cluster_3D**:

```
int * find_clusters_3D(int * binaryVector, int dim_x,
    int dim_y, int dim_z, int n, int * num_clusters)
```

restituisce la mappa dei cluster trovati utilizzando la **26-connectivity** nell'immagine binaria fornita in input.

Find connected components in binary images (2/2)

.

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 3 & 3 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 2 & 2 & 2 \\ 1 & 0 & 0 & 2 & 2 \end{bmatrix}$$

E' stato utilizzata la specifica OpenMP per rendere il calcolo degli score più veloce.

OpenMP (Open Multiprocessing) è un API multipiattaforma per la creazione di applicazioni parallele su sistemi a memoria condivisa.

Il comando:
#pragma omp parallel for
viene utilizzato per rendere un for parallelo.

Il comando:
#pragma omp atomic
invece viene utilizzato per rendere un istruzione atomica.

Abbiamo deciso di utilizzare, OMP perché l'effort per utilizzarlo è praticamente nullo, e le prestazioni sono ottime.

Inoltre essendo che l'implementazione dei *Thread* in *C* cambia tra Windows e Linux, si sarebbe reso necessario modificare il codice per renderlo funzionante su entrambe le piattaforme.