MODELOS GRÁFICOS PROBABILISTICOS

Mario De Los Santos Hernández

Proyecto 3. MRFs Optimization

Implement the stochastic simulation algorithm for obtaining the most probable configuration of a 1st order regular MRF considering the at least two variants ICM and Metropolis, using MAP

- Input: dimensions, observation Matrix
- Output: most probable configuration

Test on the image smoothing example (4 x 4 MRF) with different observations and initial configurations; vary the probabilities for Metropolis.

1. Reporte

El siguiente reporte muestra las bases para la implementación del algoritmo desarrollado, así como la referencia empleada y tomada del libro base de la clase, el cual podrán encontrar como referencia al final de este documento. Con la finalidad de facilitar el acceso al código y su replicación en caso de ser necesario se ha optado por crear un repositorio en la plataforma OpenSource GitHub, en dicho repositorio podrán encontrar las distintas versiones continuadas del proyecto, con la finalidad de continuar con la comprensión del algoritmos y los temas que engloba.

- GitHub: MRFs_Stochastic-Simulation

Algorithm 6.1 Stochastic Search Algorithm

Require: MRF, **F**; Energy function, U_F; Number of iterations, N; Number of variables, S; Probability threshold, T;
for $i = 1$ **to** S **do**
  $F(i) = l_k$ (Initialization)
end for
for $i = 1$ **to** N **do**
  for $j = 1$ **to** S **do**
    $t = l_{k+1}$ (An alternative value for variable F(i))
    if $U(t) < U(F(i))$ **then**
      $F(i) = t$ (Change value of F(i) if the energy is lower)
    else
      if $random \times \mid U(t) - U(F(i)) \mid < T$ **then**
        $F(i) = t$ (With certain probability change F(i) if the energy is higher)
      end if
    end if
  end for
end for
return $\mathbf{F}^*$ (Return final configuration)

Figura 1. Pseudocódigo del algoritmo Búsqueda Estocástica.

- Ejemplo

El código se ha desarrollado bajo el paradigma de la programación orientada a objetos, la intención de esto es poder continuar su desarrollo en un futuro, o realizar aplicaciones con otras propiedades, para ello se desarrollo una clase con las funciones necesarias para su operación, las cuales pueden apreciarse en la siguiente figura.

```
class MRF_Optimization
{
    /.../
    #define default_size_x 10
    #define default_size_y 10
    //Delimitation variables for the incoming matrixes
    int px, py;
    //Lamda value needed to calculate the energies of the "pixeles"
    //Reference: Chapter 6, pag 99. 6.4 Inference. Book in the head of the code
    int lamda_clss;
    //Number of iterations that you want to run the method selected, you can determinate any number,
    //the program would let you know when the matrix stop changing
    int iterations;

    //The following matrices are to keep the class private, adherents to the data hiding philosophy
    int Observation_mtx[default_size_x][default_size_y]={0};
    int MRF_F[default_size_x][default_size_y]={0};
    int MRF_F1[default_size_x][default_size_y]={0};

public:
    /.../
    MRF_Optimization(int size_px, int size_py, int * Obs) {...}

    /.../
    void Stochastic_ICM_MAP(int lamda, int intr) {...}
    void Stochastic_ICM_Metropolis(int lamda, int intr, float P) {...}
    ~MRF_Optimization(){}

private:
    float Smoothing(int *MRF, int row, int colum) {...}
    int Matrix_Check(int A[][default_size_y],int B[][default_size_y]) {...}

};
```

Figura 2. Estructura de la clase MRF_Optimization.

Con la intención de explicar el código, se ha estructurado en cada función el trasfondo de su uso, sin embargo, no entraremos en detalle en dichas significancias, nos enfocaremos en la aplicación del proyecto.

```
void Stochastic_ICM_MAP(int lamda, int intr)
{
    cout<<"ICM_MAP"<<endl;
    iterations  = intr;
    lamda_clss = lamda;
    float z0=0,z1=0;
    int interaction_review=0;
    int aux[default_size_x][default_size_y]={0};
    //Define a aux matrix to work with in the interactions
    for(int i=0;i<px;i++)
        for (int j = 0; j < py; j++)
            aux[i][j] = MRF_F[i][j];


    /.../
    //In MAP usually would take 1 iteraction to reconstruct the observation matrix
    for(int s=0;s<iterations +1;s++)
    {
        //Run around the rows
        for(int i=0;i<px;i++)
        {
            z0 = 0; //Rest Z after each iteraction, the function of Z0 and Z1 would be explained in the  Smoothing functions
            z1 = 0;
            //Run across the columns
            for(int j=0;j<py;j++)
            {
                //We calculate the energies for each case the 0 and 1 case for the MRF matrix in comparison with the Observation one
                z0 = Smoothing((int*)aux,i,j);
                z1 = Smoothing( MRF: (int*)MRF_F1,i,j);
                cout<<z0<<" "<<z1<<endl; //Just to confirm the decision

                //Base selection, we would take the option with less energy.
                /.../
                if(z0>=z1)
                    MRF_F[i][j] = MRF_F1[i][j];
            }
        }
        //The matrix check function, would basically confirm if the two matrix are the same. If is the case, we would have a 1 in the variable
        iteraction_review = Matrix_Check(MRF_F,Observation_mtx);

        // If Iteraction review is 1, we return the number of iteractions needed ant then brake the iteractions loop
        if(iteraction_review == 1) {
            cout<<"Finished in: "<<s<<" interactions"<<endl;
            break;}
    }
  //For this version we use the vizualization just to confirm the results, the future development in this code is review in the head of the code
  cout<<"Vizualization, just debbuging"<<endl;
    for(int i=0;i<px;i++) {
        for (int j = 0; j < py; j++) {
            cout<<(MRF_F[i][j]);}
    cout<<" "<<endl;
    }
}
```

Figura 3. Algoritmo de búsqueda estocástica bajo la variable ICM-MAP.

```
void Stochastic_ICM_Metropolis(int lamda, int intr, float P)
{
    cout<<"Metropolis"<<endl;
    iterations  = intr;
    lamda_clss = lamda;
    float z0=0,z1=0;
    int iteraction_review=0;
    int aux[default_size_x][default_size_y]={0};

    //Define a aux matrix to work with in the interactions
    for(int i=0;i<px;i++)
        for (int j = 0; j < py; j++)
            aux[i][j] = MRF_F[i][j];

    //Run around the iteractions
    for(int s=0;s<iterations ;s++)
    {
        //Run across the rows
        for(int i=0;i<px;i++)
        {
            z0 = 0; //Rest Z after each iteraction, the function of Z0 and Z1 would be explained in the  Smoothing functions
            z1 = 0;
            //Run across the columns
            for(int j=0;j<py;j++)
            {
                //We calculate the energies for each case the 0 and 1 case for the MRF matrix in comparison with the Observation one
                z0 = Smoothing((int*)aux,i,j);
                z1 = Smoothing( MRF: (int*)MRF_F1,i,j);                    //Base selection, we would take the option with less energy.
                /.../

                if(z0>=z1) {
                    float uf = float(rand() % (11027 + 1)) / 11027;
                    cout<<uf<<endl;
                    if(uf>P)
                        MRF_F[i][j]=MRF_F1[i][j];
                    /.../
                }
            }
        }
        //The matrix check function, would basically confirm if the two matrix are the same. If is the case, we would have a 1 in the variable
        iteraction_review = Matrix_Check(MRF_F,Observation_mtx);

        // If Iteraction review is 1, we return the number of iteractions needed ant then brake the iteractions loop
        if(iteraction_review == 1) {
            cout<<"Finished in: "<<s<<" interactions"<<endl;
            break;}
    }
    //For this version we use the vizualization just to confirm the results, the future development in this code is review in the head of the code
    for(int i=0;i<px;i++) {
        for (int j = 0; j < py; j++) {
            cout << MRF_F[i][j];
            MRF_F[i][j] = 0;
        }
        cout<<endl;
    }
}
```

Figura 4. Algoritmo de búsqueda estocástica bajo la variable ICM-Metrópolis.

El uso de la clase se ha probado con el ejercicio realizado como tarea para el curso, el cual es el siguiente, considerando que para esa aplicación se ha hecho de manera manual, los resultados en el caso de Metrópolis variaran en relación a la generación de números aleatorios que implica el método.

	0	0	0	0
F	0	0	0	0
	0	0	0	0
	0	0	0	0

Lambda 4

	0	0	0	0
G	0	1	1	0
	0	1	0	0
	0	0	1	0

V1(0)	V1(1)
0	6
V5(0)	V5(1)
0	7
V9(0)	V9(1)
0	3
V13(0)	V13(1)
0	6

V2(0)	V2(1)
0	7
V6(0)	V6(1)
4	3
V10(0)	V10(1)
4	4
V14(0)	V14(1)
0	6

V3(0)	V3(1)
0	7
V7(0)	V7(1)
4	3
V11(0)	V11(1)
0	8
V15(0)	V15(1)
4	4

V4(0)	V4(1)
0	6
V8(0)	V8(1)
0	6
V12(0)	V12(1)
0	7
V16(0)	V16(1)
0	6

MAP	0	0	0	0
	0	1	1	0
	0	1	0	0
	0	0	1	0

Figura 5. Tarea usada como ejemplo.

```
#define px 4 //Matrix's sizee
#define py 4//Matrix's size

int Observation_matrix[px][py]={{0,0,0,0}, //Observation Matrix for testing
                                {0,1,1,0},
                                {0,1,0,0},
                                {0,0,1,0}};
int lamda = 4; //Lamba constant needed to calculate the energies
int No_interactions=1; //Number of interactions for both process ICM MAP and ICM Metropolis
float Metropolis_Prob = 0.5; //Probability value

MRF_Optimization T1(px, py, (int*)Observation_matrix); //Object creation, read the description in the class
```

Figura 6. Definición del ejercicio, considere que G es la matriz de observación.

Para el llamado de los métodos utilizamos las siguientes funciones. Los parámetros para cada función dependen de su método, por ejemplo para el caso de ICM-MAP, solo necesitamos lambda y el número de iteraciones.

```
//ICM-MAP
T1.Stochastic_ICM_MAP(lamda,No_interactions);
//Metropolois
T1.Stochastic_ICM_Metropolis(lamda,No_interactions,Metropolis_Prob);
```

Figura 7. Llamado de funciones.

```
ICM_MAP
0 6
0 6
0 6
0 6
0 7
4 4
5 3
1 6
0 7
5 3
2 6
0 7
0 6
0 7
4 3
1 5
Finished in: 0 interactions
Vizualization, just debbuging
0000
0110
0100
0010
```

Figura 8. ICM-MAP resultado, considere que observamos los resultados de las energías en cada caso.

```
Metropolis
0.00371815
0.674617
0.574408
0.403011
0000
0010
0100
0000
```

Figura 9. Resultado de Metrópolis en una sola iteración.

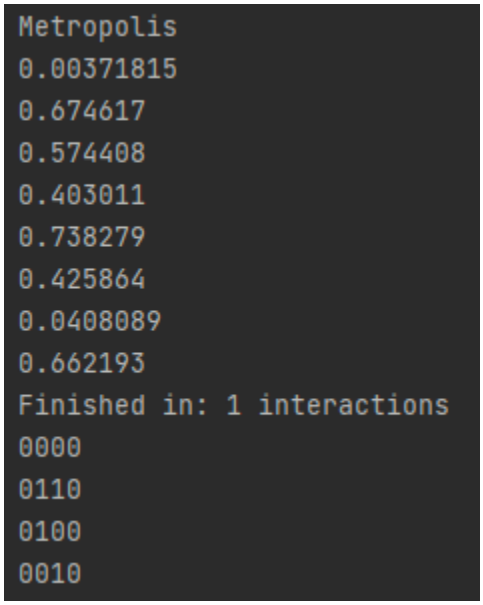

Figura 10. Resultado de Metrópolis, corrida alterna.

Recordemos en el caso de Metrópolis que nos basamos en la generación de números aleatorios para poder tomar una decisión respecto a la energía a tomar.

2. Referencias

a. Sucar, L. E. (2020). Probabilistic graphical models. Advances in Computer Vision and Pattern Recognition.London: Springer London. doi, 10(978), 2.