UNIVERSIDAD DE GUANAJUATO

DIVISIÓN DE INGENIERÍAS

CAMPUS – IRAPUATO SALAMANCA

Inteligencia Artificial

PROF. GARCÍA CAPULÍN CARLOS HUGO

TAREA 6

Sintonización del algoritmo PSO

Martínez Lona Verónica Montserrat

Martes 7 Marzo, 2017

Introducción:

Hicimos la sintonización del algoritmo PSO y sus efectos en el desempeño. También analisamos las ventajas o desventajas de aplicar alguno de las opciones de sintonización.

La sintonización sel algoritmo se constituye de:

- Número de parámetros de las partículas.
- Número de partículas en el emjambre.
- Límites para el espacio de búsqueda de las partículas.
- Condicón de término (Número de iteraciones / Umbral de error).

Desarrollo:

Ya teniendo el algoritmo desarrollado nos dedicamos a hacer pruebas simples de rendimiento para ver de qué manera afectaba cambiar ciertos parámetros en el algoritmo, tales como el número de partículas o el margen de error.

Los números que escogimos y los límites fueron meramente sin pensar y en ocasiones algo demasiado grande para observar claramente el efecto que tenía sobre la eficiencia del método.

El desarrollo de la práctica consistió básicamente en modificar los siguientes valores:

const unsigned int NUMBER_PARTICLES;
const unsigned int NUMBER_ITERATIONS;
const float LimiteInf;
const float LimiteSup;
const float Vmin;
const float Vmax;
(50 - expSwarm -> particle[expSwarm -> idGbest].PFit) > error;

Código:

```
//Desarrollo de PSO
/*
Problema de ejemplo:
  Maximizar la función:
    f(x, y) = 50 - (x-5) ^ 2 + (y - 5) ^ 2
*/
#include<stdio.h>
#include<stdlib.h>

//Particle definition
typedef struct {
  float *Xi; //Actual Position
  float *Vi; //Actual Speed
  float *Pi; //Best Position
  float XFit; //Fitness value
  float PFit; //Best position fitness value
} PARTICLE;

//Swarm definition
typedef struct {
  unsigned int numParticles; //Number of particles of the swarm
  unsigned int numParameters; //Number of parameters of the equation
  unsigned int idGbest; //id of the best particle
  float Vmin;
  float Vmax;
  float c1; //Random constant
  float c2; //Random constant
  PARTICLE *particle; //swarm
} SWARM;

const unsigned int NUMBER_PARTICLES = 80;
const unsigned int NUMBER_PARAMETERS = 2;
const unsigned int NUMBER_ITERATIONS = 200;
const float LimiteInf = 0;
const float LimiteSup = 10;
const float C1 = 2; //por teoria toman este valor
const float C2 = 2;
const float Vmin = -1.0;
const float Vmax = 1.0;

PARTICLE *CreateParticle(void);
SWARM *CreateSwarm(const unsigned int numParticles, const unsigned int numParameters);
void InitSwarm(SWARM *pSwarm, const float infL, const float supL,  const float Vmin, const float Vmax, const float c1, const float c2);
void InitBest(SWARM *pSwarm);
void UpdateSpeed(SWARM *pSwarm);
void UpdatePosition(SWARM *pSwarm);
void UpdateBest(SWARM *pSwarm);
void EvaluateSwarm(SWARM *pSwarm);
void DeleteSwarm(SWARM *pSwarm);
void ShowParticle(SWARM *pSwarm, const unsigned int i);
void ShowSwarm(SWARM *pSwarm);

int main(void)
{
  SWARM *expSwarm;
  unsigned int it = 0;
  unsigned int maxIt = NUMBER_ITERATIONS;
  //Crear memoria para el enjambre
  expSwarm = CreateSwarm(NUMBER_PARTICLES, NUMBER_PARAMETERS);
```

```
 //Inicializar posicones de las partículas ente los límite del espacio de búsqueda del problema
 InitSwarm(expSwarm, LimiteInf, LimiteSup, Vmin, Vmax, C1, C2);
 //Iniciar el fitness de cada partícula e identificar el fitness de la mejor global
 InitBest(expSwarm);

 while((it < maxIt) && (50 - expSwarm -> particle[expSwarm -> idGbest].PFit) > 0.00001)
 {
  //Evaluar las partículas del enjambre en la ecuacion
  EvaluateSwarm(expSwarm);
  UpdateSpeed(expSwarm);
  UpdatePosition(expSwarm);
  UpdateBest(expSwarm);
  //Mostrar todas las partículas del enjambre
  ShowSwarm(expSwarm);
  printf("idGbest: %d \n", expSwarm -> idGbest);
  printf("it = %d ----------\n", it);
  it++;
 }

 //Liberar la memoria del enjambre
 DeleteSwarm(expSwarm);

 return 0;
}

SWARM *CreateSwarm(const unsigned int numParticles, const unsigned int numParameters)
{
 SWARM *pSwarm;

 //_____Asignar memoria para la estructura enjambre
 pSwarm = (SWARM*)malloc(sizeof(SWARM));
 if (pSwarm == NULL)
 {
  printf("Error al asignar memoria a la estructura enjambre.\n");
  return -1;
 }
 //_____Inicializar la estructura
 //Asignar número de partículas
 pSwarm -> numParticles = numParticles;
 //Asignar memoria para las partículas
 pSwarm -> particle = (PARTICLE*)malloc(numParticles*sizeof(PARTICLE));
 if(pSwarm -> particle == NULL)
 {
  printf("Error al asignar memoria a las particulas.\n");
  return -1;
 }
 pSwarm -> numParameters = numParameters;
 //Asignar la memoria a cada partícula
 for (unsigned int i = 0; i < numParticles; i++)
 {
  pSwarm -> particle[i].Xi = (float*)malloc(numParameters*sizeof(float));
  pSwarm -> particle[i].Vi = (float*)malloc(numParameters*sizeof(float));
  pSwarm -> particle[i].Pi = (float*)malloc(numParameters*sizeof(float));
 }

 return pSwarm;
}

void InitSwarm(SWARM *pSwarm, const float infL, const float supL,  const float Vmin, const float Vmax, const float c1, const float c2)
{
 unsigned int i, j;
 double r;
```

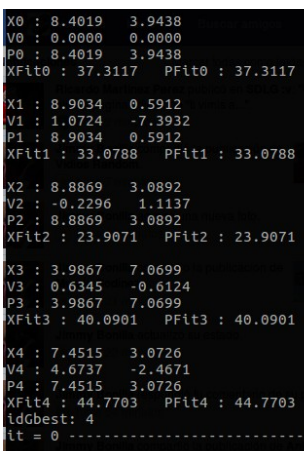
```
 pSwarm -> c1 = c1;
 pSwarm -> c2 = c2;

 pSwarm -> Vmin = Vmin;
 pSwarm -> Vmax = Vmax;

 //Para todas las particulas
 for(i = 0; i < pSwarm -> numParticles; i++)
 {
  //Para todos los parametros
  for(j = 0; j < pSwarm -> numParameters; j++)
  {
   r = ((double) rand() / RAND_MAX) * (supL - infL) + infL; //Valor random entre InfL-SupL
   //Posicion aleatoria
   pSwarm -> particle[i].Xi[j] = r;
   //Asignamos velocidad 0, pues aun no se empiezan a mover
   pSwarm -> particle[i].Vi[j] = 0;
   //Peso aleatorio
   pSwarm -> particle[i].Pi[j] = r;
  }
   pSwarm -> particle[i].XFit = 0;
   pSwarm -> particle[i].PFit = 0;
 }


 return;
}

void InitBest(SWARM *pSwarm)
{
 unsigned int i;
 float best;
 best = 0;

 for(i = 0; i < pSwarm -> numParticles; i++)
 {
  //En la inicalización el mejor peso es directamente el peso de la particula, pues no hay mas en el historico
  pSwarm -> particle[i].PFit = pSwarm -> particle[i].XFit;
  if(pSwarm -> particle[i].XFit > best)
  {
     //Comparar los pesos de las particulas del ejambre poara saber cual de ellas es la mejor
     pSwarm -> idGbest = i;
     best = pSwarm -> particle[i].PFit;
  }
 }

 return;
}

void EvaluateSwarm(SWARM *pSwarm)
{
 //Evaluamos las particulas en la ecuacion
 unsigned int i;
 for(i = 0; i < pSwarm -> numParticles; i++)
 {
  //f(x,y) = (x-5)°2 + (y-5)ª2
  pSwarm -> particle[i].XFit = 50 - ((pSwarm -> particle[i].Xi[0] - 5) * (pSwarm -> particle[i].Xi[0] - 5) +
                        (pSwarm -> particle[i].Xi[1]  -5 ) * (pSwarm -> particle[i].Xi[1] - 5));
 }
 return;
}
```

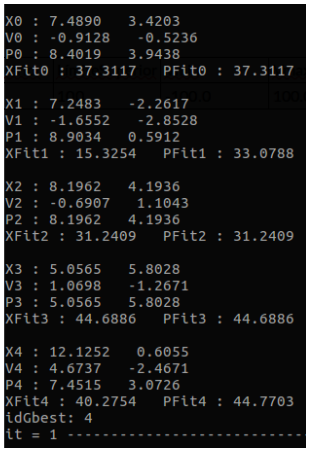
```
void UpdateSpeed(SWARM *pSwarm)
{
 unsigned int i, j;
 float y1, y2, aux;

srand(time(NULL));

 //Para todas las partículas
 for(i = 0; i < pSwarm -> numParticles; i++)
 {
  //para todos los parámetros
  for(j = 0; j < pSwarm -> numParameters; j++)
  {
   //Variables aleatorias
   y1 = ((double)rand()/RAND_MAX);
   y2 = ((double)rand()/RAND_MAX);
   //Càlculo de la formula que marca el algoritmo
   aux = pSwarm -> particle[i].Vi[j] + pSwarm -> c1 * y1 * (pSwarm -> particle[i].Pi[j] - pSwarm -> particle[i].Xi[j]) +
                          pSwarm -> c2 * y2 * (pSwarm -> particle[pSwarm -> idGbest].Pi[j] - pSwarm -> particle[i].Xi[j]);
   if(aux > pSwarm -> Vmax)
   {
    pSwarm -> particle[i].Vi[j] = pSwarm -> Vmax;
    continue;
   }
   if(aux < pSwarm -> Vmin)
   {
    pSwarm -> particle[i].Vi[j] = pSwarm -> Vmin;
    continue;
   }

   pSwarm -> particle[i].Vi[j] = aux;
  }
 }

 return;
}

void UpdatePosition(SWARM *pSwarm)
{
 unsigned int i, j;

 //para todas las particulas
 for(i = 0; i < pSwarm -> numParticles; i++)
 {
  //para todos los parametros
  for(j = 0; j < pSwarm -> numParameters; j++)
  {
   //posicion actual mas la velocidad
   pSwarm -> particle[i].Xi[j] += pSwarm -> particle[i].Vi[j];
  }
 }

 return;
}

void UpdateBest(SWARM *pSwarm)
{
 unsigned int i, j;
 //Peso de la mejor partícula del enjambre
 float best = pSwarm -> particle[pSwarm -> idGbest].PFit;
```

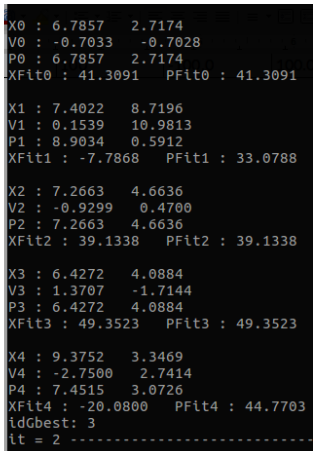
```
 //Para todas las particulas
 for(i = 0; i < pSwarm -> numParticles; i++)
 {
  //Si el peso de actual es mayor que el del historico de esa particula
  if(pSwarm -> particle[i].XFit > pSwarm -> particle[i].PFit)
  {
   //Para todos los parametros
   for(j = 0; j < pSwarm -> numParameters; j++)
   {
    //Se actualizan los pesos optimos
    pSwarm -> particle[i].Pi[j] = pSwarm -> particle[i].Xi[j];
    pSwarm -> particle[i].PFit = pSwarm -> particle[i].XFit;
   }
  }
  //Si el mejor peso actual de la particula es mejor que el de la mejor del ejambre
  if(pSwarm -> particle[i].PFit > best)
  {
   //Se actualiza el id y peso del mejor
   pSwarm -> idGbest = i;
   best = pSwarm -> particle[i].PFit;
  }
 }

 return;
}

void ShowParticle(SWARM *pSwarm, const unsigned int i)
{
 unsigned int j;

 printf("\nX%d : ", i);
 for(j = 0; j < pSwarm -> numParameters; j++)
 {
  printf("%2.4f   ", pSwarm -> particle[i].Xi[j]);
 }
 printf("\nV%d : ", i);
 for(j = 0; j < pSwarm -> numParameters; j++)
 {
  printf("%2.4f   ", pSwarm -> particle[i].Vi[j]);
 }
 printf("\nP%d : ", i);
 for(j = 0; j < pSwarm -> numParameters; j++)
 {
  printf("%2.4f   ", pSwarm -> particle[i].Pi[j]);
 }
 printf("\nXFit%d : ", i);
 printf("%2.4f   ", pSwarm -> particle[i].XFit);

 printf("PFit%d : ", i);
 printf("%2.4f\n", pSwarm -> particle[i].PFit);

 return;
}

void ShowSwarm(SWARM *pSwarm)
{
 unsigned int i;
 //Para todas las partículas
 for(i = 0; i < pSwarm -> numParticles; i++)
 {
  ShowParticle(pSwarm, i);
 }
```

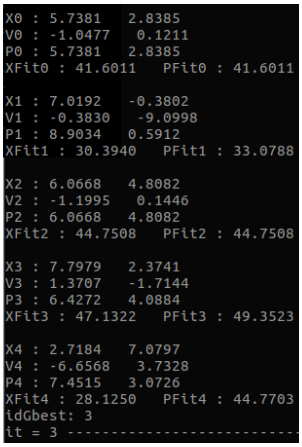
```
  return;
}

void DeleteSwarm(SWARM *pSwarm)
{
  unsigned int i;

  for (i = 0; i < pSwarm -> numParticles; i++)
  {
    free(pSwarm -> particle[i].Xi);
    free(pSwarm -> particle[i].Vi);
    free(pSwarm -> particle[i].Pi);
  }

  free(pSwarm);

  return;
}
```

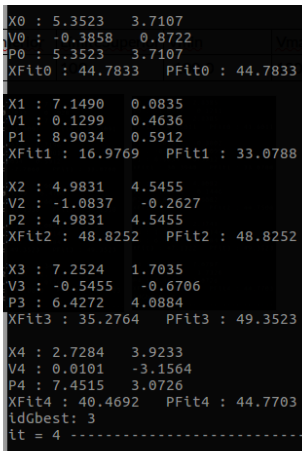
Pruebas y resultados

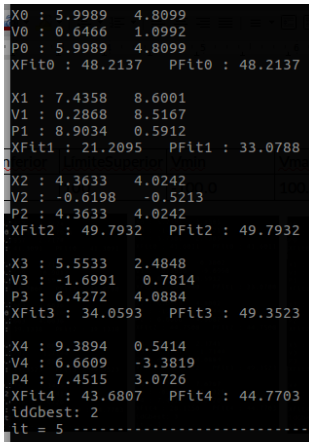


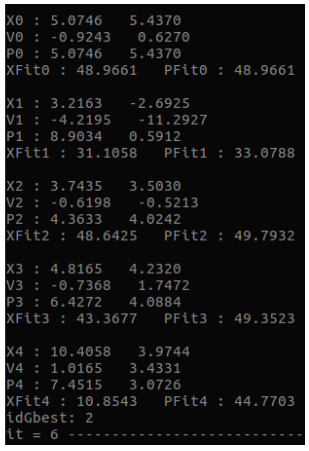


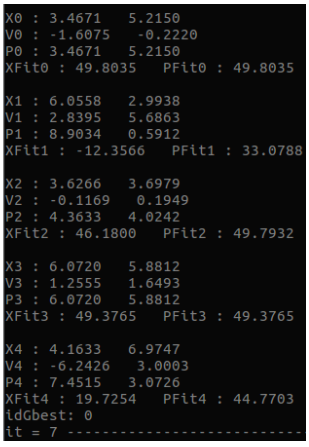


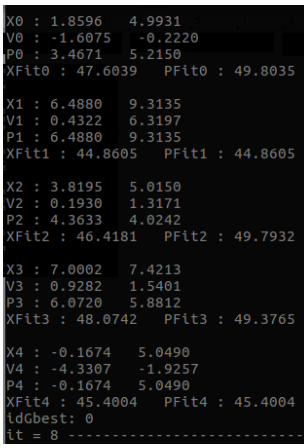


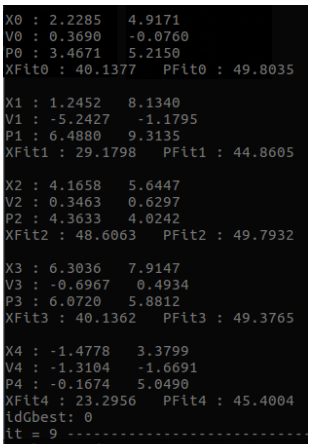












Las primeras pruebas fueron cambiando el número de partículas y viendo cómo este cambio afectaba el peso de la solución encontrada. La conclusión fue que **A MAYOR NÚMERO DE PARTÍCULAS MEJORA EL RESULTADO.** Este número depende de la aplicación y se puede determinar de forma experimental, aunque los resultados generals marcan un rango de 20 a 80 partículas poor enjambre en la mayoría de los problemas.

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	10	0	10	-100.0	100.0	49.8035

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
20	10	0	10	-100.0	100.0	49.9074

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
50	10	0	10	-100.0	100.0	49.9760

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
100	10	0	10	-100.0	100.0	49.9865

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
200	10	0	10	-100.0	100.0	49.9982

Se experimentó también cambiando el rango de búsqueda de la solución. La conclusión aquí fue que **UN RANGO PEQUEÑO Y NO MUY ALEJADO DE LA SOLUCIÓN FUNCIONA MEJOR.** La desvebtaja de acotar el rango de búsqueda es que no siempre se conoce la solución, entonces existe la posibilidad de que dejemos a la solución fuera del rango de búsqueda, por lo que este parámetro debe ser utilizado con cuidado.
Otra cosa que se pudo observar es que las prtículas se perdían con un rango muy grande y terminaban muy lejos de la solución o incluso no se podían encontrar.

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	10	0	10	-100.0	100.0	49.8139

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	10	-10	10	-100.0	100.0	49.7167

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	10	-20	20	-100.0	100.0	49.1216

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	10	-50	50	-100.0	100.0	Indef

También se experimentó con el número de iteraciones y se pudo observar que **NO SIEMPRE MÁS ITERACIONES SIGNIFICA ESTAR MÁS CERCA DE LA SOLUCIÓN.** Para mi caso en particular 100 iteraciones fue lo que más me acercó a la solución.

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	10	0	10	-100.0	100.0	49.9731

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	20	0	10	-100.0	100.0	49.0602

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	50	0	10	-100.0	100.0	49.3008

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	100	0	10	-100.0	100.0	49.9991

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	200	0	10	-100.0	100.0	49.9823

La velocidad es un parámetro que usualmente se controla, pues después de cierto tipo de iteraciones las partículas suelen explorar y alejarse demasiado de la solución. Lo que pude concluir es que **LIMITAR LA VELOCIDAD AYUDA A LLEGAR A LA SOLUCIÓN DE UNA MANERA MÁS RÁPIDA.**

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	10	0	10	-100.0	100.0	49.2789

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	10	0	10	-50.0	50.0	49.9830

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	10	0	10	-20.0	20.0	49.9889

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	10	0	10	-5.0	5.0	49.9483

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	10	0	10	-1.0	1.0	49.9722

Para ahorrar algunas iteraciones se pone el parámetro de error, que sirve para que cuando encontremos una solución que cumpla el error la ejecución se detenga ahorrando tiempo de cómputo si el programa lo encuentra antes de agotar el límite de iteraciones.

Particles	Iterations	Error	LímiteInferior	LímiteSuperior	Vmin	Vmax	Iteration
5	100	1	0	10	-1.0	1.0	2

Particles	Iterations	Error	LímiteInferior	LímiteSuperior	Vmin	Vmax	Iteration
5	100	0.5	0	10	-1.0	1.0	17

Particles	Iterations	Error	LímiteInferior	LímiteSuperior	Vmin	Vmax	Iteration
5	100	0.1	0	10	-1.0	1.0	4

Particles	Iterations	Error	LímiteInf	LímiteSup	Vmin	Vmax	Iteration
5	100	0.001	0	10	-1.0	1.0	99

Particles	Iterations	Error	LímiteInf	LímiteSup	Vmin	Vmax	Iteration
5	100	0.0001	0	10	-1.0	1.0	99

Las últimas pruebas se realizaron utilizando los parámetros que resultaron óptimos para mi implementación del algoritmo y registrando en qué iteración alcanza una solución con cierto margen de error. Observando los resultados se puede observar que llega de una manera mucho más rápida a la solución, ahorrando iteraciones y por tanto tiempo de cómputo.

Particles	Iterations	Error	LímiteInferior	LímiteSuperior	Vmin	Vmax	Iteration
200	100	1	0	10	-1.0	1.0	0

Particles	Iterations	Error	LímiteInferior	LímiteSuperior	Vmin	Vmax	Iteration
200	100	0.5	0	10	-1.0	1.0	1

Particles	Iterations	Error	LímiteInferior	LímiteSuperior	Vmin	Vmax	Iteration
200	100	0.001	0	10	-1.0	1.0	16

Particles	Iterations	Error	LímiteInferior	LímiteSuperior	Vmin	Vmax	Iteration
200	100	0.0001	0	10	-1.0	1.0	65

Conclusión

Incluso despuès de haber implementado el algortimo se deben definir cuertos paràmetros o constantes para hacer que tenga el mejor rendimiento. Estos parámetros pueden definirse de forma experimental, pero de igual modo ya hay cifras determinadas que suelen funcionar para la mayor parte de las aplicaciones. De cualquier modo es bueno revisar cómo influyen estos parámetros en nuestra implementación particular, pues la eficiencia depende de muchos factores muy particulas (SO, hardware, recurrencias, ciclos, etc), que hacen que ciertos números funcionen mejor que lo que la teoría dicta.

Siempre es bueno experimentar un poco con los algortimos antes de inicar a aplicarlos, pues podemos entenderlos mejor y saber qué acomodos hacerle para que funcione de una manera mejor, màs eficiente y rápida.